Università di Bologna · Universität Bielefeld

School of Science · Faculty of Technology

Computer Science · Intelligent Systems

Deep learning based style transfer for low altitude aerial imagery

Supervisor:
Prof. Barbara Hammer
Prof. Serena Morigi
Kai Konen, M.Sc.

Candidate:
Federico Pennino, B.Sc.

Second session
Academic year 2021/2022

Abstract

Unmanned Aerial Vehicle (UAVs) equipped with cameras have been fast deployed to a wide range of applications, such as smart cities, agriculture or search and rescue applications. Even though UAV datasets exist, the amount of open and quality UAV datasets is limited. So far we want to overcome this lack of high quality annotation data by developing a simulation framework for a parametric generation of synthetic data. The framework accepts input via a serializable format. The input specifies which environment preset is used, the objects to be placed in the environment along with their position and orientation as well as additional information such as object color and size. The result is an environment that is able to produce UAV typical data: RGB image from the UAVs camera, altitude, roll, pitch and yawn of the UAV. Beyond the image generation process, we improve the resulting image data photorealism by using Synthetic-to-real transfer learning methods. Transfer learning focuses on storing knowledge gained while solving one problem and applying it to a different - although related - problem. This approach has been widely researched in other affine fields and results demonstrate it to be an interesing area to investigate. Since simulated images are easy to create and synthetic-to-real translation has shown good quality results, we are able to generate pseudo-realistic images. Furthermore object labels are inherently given, so we are capable of extending the already existing UAV datasets with realistic quality images and high resolution meta-data. During the development of this thesis we have been able to produce a result of 68.4% on UAVid. This can be considered a new state-of-art result on this dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays the usage of UAVs is spreading in the *Machine Learning* community. They have been used in a wide range of applications showing interesting results in most of them. Chowdhury *et al.* showed how UAVs [11] could be applied to improve the existing methodologies for natural disaster damage assessment. The authors provide images from both affected and non-affected areas after hurricane *Michael*, and tested them with different semantic segmentation methods. On the other hand, in [50] UAVs imagery is used in an agricultural environment. Rahul Raj *et al.* use UAV-based Precision Agriculture - an approach to farm management that uses information technology for optimizing health and productivity - to identify stressed areas, and help to carried out some corrective measures (e.g., fertilizer and pesticide spraying). However, general purpose open-source datasets with a good quality of annotations and coherent annotations are missing. This is a huge limitation because at UAVs altitude the performance of computer vision algorithms is inevitably limited by illumination and visual pollution; so a large amount of data is needed for training. In [78] there is an example of general purpose UAV dataset. It is composed of 8k images with more than 540k bounding boxes, but information as altitude, yaw or pitch - as well as semantic segmentation or instance segmentation - are not provided. Another example of existing UAV dataset is presented in [41] where information - as altitude, yaw or pitch - are still missing, but semantic segmentation is provided. These problems are the reasons why *Machine Learning* community is shifting its interest on synthetic datasets. Synthetic datasets allow for easily generated high resolution images with out-of-box good quality annotations. Kiefer, Ott and Zell [30] provide a synthetic UAV dataset example. They have extended DeepGTAV to work for UAV scenarios and they capture various synthetic datasets in several domains. In the same direction, VALID dataset [7] is an example of synthetic dataset made from scratch. Authors provide the first aerial image dataset that can provide panoptic level segmentation and complete dense depth maps. It is important to note how usage of synthetic datasets like VALID for UAVs has shown good improvements on Computer Vision algorithms. Konen and Hecking [31] worked on the direction of *ANN*s (Artificial

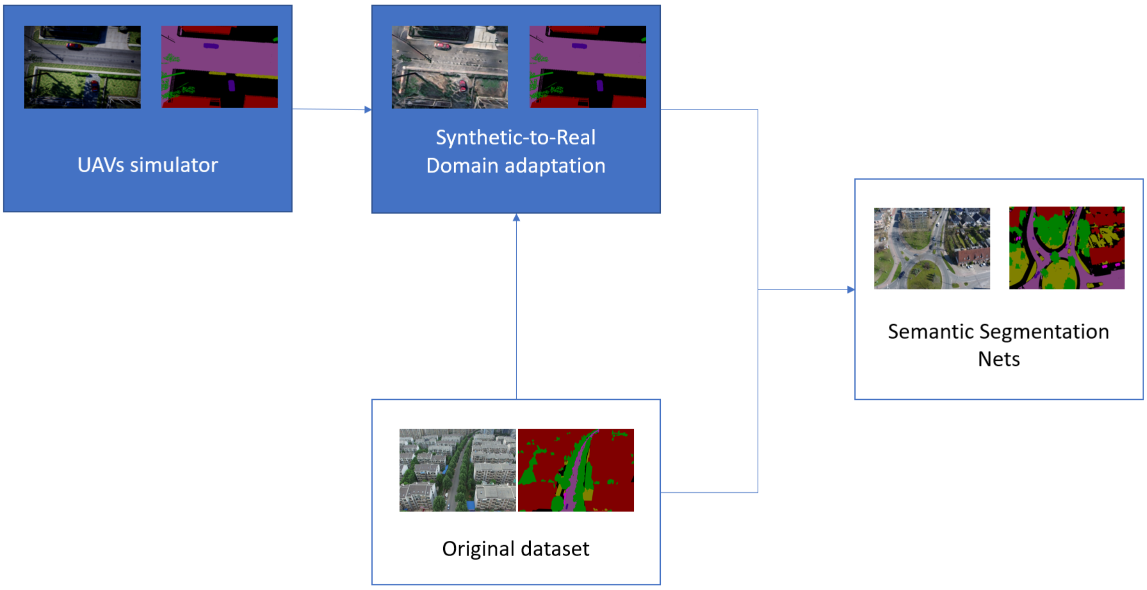


Figure 1.1: Visual representation of the work we made during this thesis. Blue blocks are the parts where we have focused on.

Neural Networks) training in UAVs context using synthetic data generated with AirSim [59] for data augmentation. They have shown how synthetic data can help to increase robustness of object detection model. Nevertheless, the usage of this kind of data is still limited because only low resolution meta-data are provided. The goal we want to pursue with this thesis is to overcome this lack of high quality annotation data by developing a simulation framework for a parametric generation of synthetic data. Beyond the data generation process, we explore the possibilty to augment image photorealism by using Synthetic-to-real transfer learning methods. Transfer learning [45] is a research problem that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. We use GANs (Generative Adversarial Networks)[28] model to improve the realism of generated images. This approach has been widely researched in other affine fields so we translate it to our domain of interest. In Deep CG2Real[4] the authors improve the visual realism of low-quality synthetic images in OpenGL renderings. This work shows how the synthetic-to-real translation can improve the final quality render. Another similar work is [51] where Richter, AlHaija and Koltun present an approach to enhance the realism of synthetic images. The images are enhanced by a convolutional network that leverages intermediate representations produced by conventional rendering pipelines. It is shown how this approach can impressively increase the realism of generated images. Since simulated images are easy to create and synthetic-to-real translation has shown improvements on the image quality, we are able to generate realistic quality images. Furthermore object labels are inherently given, so we are capable of extending the already existing UAV datasets with realistic quality images and high resolution meta-data. The result is then a framework able to generate realistic looking imagery with out of the box label and high resolution metadata. We trained *PSPNet*, *Fully Convolutional Network* and *DeepLabV3+* to test our results. The goal of these networks is semantic segmentation. In a first instance we tested our synthetic images using a dataset obtained via various other UAV datasets. We called our final dataset REAL. We used this dataset even to perform domain-adaptation on *CycleGAN*. We get an mIoU value of $\approx 70\%$ on it. Then, we used UAVid [38] - a 4K UAVs imagery dataset - as benchmark. Changing the preprocessing strategy and using synthetic data - from simulator - we have been able to out-perform the original paper results of approximately a 20%. This allow us to out-perform the current state-of-art on this dataset. In general - despite the domain gap - original simulation images have shown to be more helpful than *CycleGAN* generated ones. A visual representation of our pipeline is shown in Figure 1.1. We took images from the simulator and shifted them to the real domain. Then we combined original images and synthetic ones to train semantic segmentation networks.

In chapter 2 we discuss some theoretical concepts that are at the basis of this thesis. Besides, chapter 3 is used to present some works that deserve to be mentioned for a better understanding in the reading of this thesis. In chapter 4 we present our simulator. We are going to discuss what is the idea behind it and how we modeled it. This chapter is the kernel of the thesis. In chapter 5 we are going to present the work we have done to improve the photo-realism of the simulator images. In chapter 6 we are going to present some experiments that lead us to the results we are going to show in chapter 7. In conclusion, in chapter 8 we are going to briefly discuss the future of this work and how we can improve it.

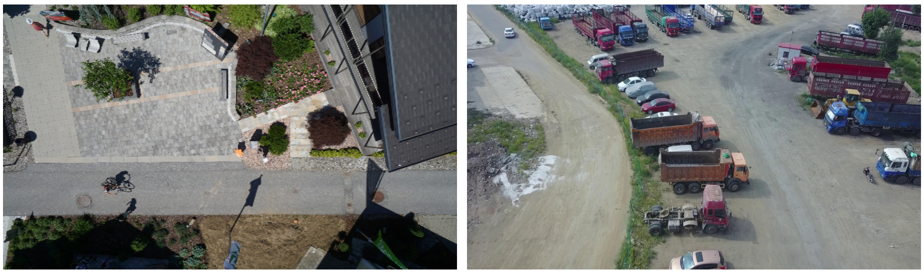
Chapter 2

Background

In this chapter, we will introduce the concepts behind this thesis. First, section 2.1 is used to introduce what UAVs are, and we show their use cases. Following that, we will discuss the segmentation task and which networks we decided to use to perform it (section 2.2). In section 2.3 we present what GAN (*Genearative Adversarial Network*) are and what the idea behind them is. We are going to give an overview of GAN metrics in section 2.3.1. Finally, in section 2.4 we give an overview of the current state-of-art on the task of Synthetic-To-Real Domain Adaptation.

2.1 UAVs

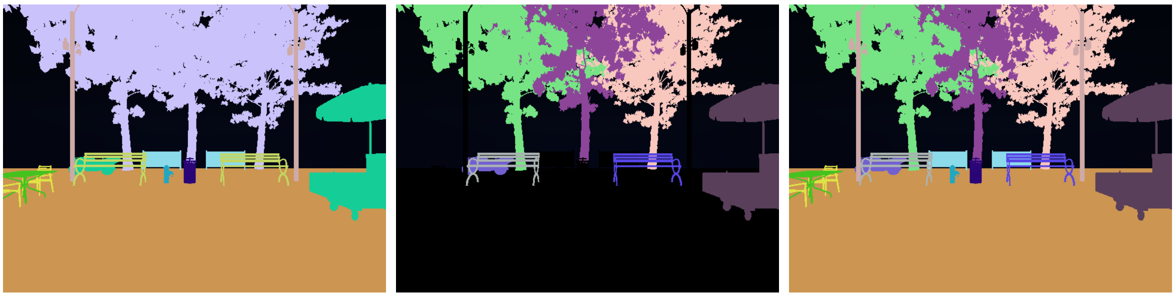
Unmanned aerial vehicles (UAV) are a class of air crafts that can fly without the onboard presence of pilots [33]. Based on their maximum altitude and maximum range, UAVs can be classified into three classes: small, medium, and large. The maximum altitude of small drones is below 300 m, while large ones exceed 5500 m. The altitude of medium ones is between these ranges. Altitude is not the only necessary information for UAVs;



(a) Semantic drone dataset

(b) VisDrone Challenge 2019

Figure 2.1: Example of images taken via UAVs.



(a) Semantic (b) Instance (c) Panoptic

Figure 2.2: A visual example of differences between semantic segmentation, instance segmentation, and panoptic segmentation. Images are taken from our simulator.

others are roll, pitch, yaw, speed, and acceleration.
In the last few years, their usage has spread in several domains because, in civil infrastructure, they allow for reduced risks and costs. We focus on those UAVs equipped with cameras and with the ability to take pictures. Our task is to perform semantic segmentation and object detection over these kinds of images. We are not focused on a specific context, but this work can be easily extended to SAR (Search And Rescue) or real-time monitoring of road traffic. In [58], an extensive introduction to the usage and research field for UAVs is given. Authors range computer vision compatible tasks to communications and wireless tasks. An example of images captured from UAVs is shown in Figure 2.1.

2.2 Image segmentation

Image classification, object detection, and semantic segmentation are all closely related tasks in Deep Learning. All three tasks involve learning the features of the image and using the learned features in various other tasks like image captioning and annotation. Specifically, segmentation is the process of partitioning a digital image into multiple image segments, also known as image regions or objects (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something more meaningful and easier to analyze. Current image segmentation methods can be classified into three categories:

- Semantic segmentation [40], objects shown in an image are grouped based on defined categories;
- Instance segmentation [23], requires the detection of multiple instances of different objects present in an image;

- Panoptic Segmentation [16], is a hybrid method combining semantic segmentation and instance segmentation.

A visual comparison is shown in Figure 2.2. These tasks are connected: they are just representing different levels of semantic annotations. As shown, semantic segmentation is just a form of pixel-level classification where each pixel in an image is classified according to a category. Therefore this segmentation can be at a different level of grouping. Then our goal is to take an RGB color image and output a segmentation map where each pixel contains a class label represented as an integer. Applications for semantic segmentation include road segmentation for autonomous driving and cancer cell segmentation for medical diagnosis. We used MMSegmentation [12] - an open source semantic segmentation toolbox based on PyTorch - to perform the evaluation. We tried out three different architectures: Fully Convolutional Network (section 2.2.1), PSPNet (section 2.2.2) and DeepLabv3+ (section 2.2.3).

2.2.1 Fully Convolutional Network

A fully convolution network (FCN) [37] is a neural network that only performs convolution, subsampling, and upsampling. They are an architecture used mainly for semantic segmentation. Usually, the network consists of a downsampling path, used to extract and interpret the context, and an upsampling path, which allows for localization. In this way, "fully convolutional" networks take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning.

The typical convolution neural network (CNN) [44] is not fully convolutional because it often contains fully connected layers. However, fully connected layers can also be viewed as convolutions with kernels that cover the entire input regions, which is the main idea behind converting a CNN to an FCN.

Generally, while the fully connected layer of an image classification net (e.g. AlexNet [32]) completely discards spatial information and delivers only one feature vector for the entire image, the fully convolutional layer produces a feature vector for every pixel. Based on this feature map, a pixel-wise classification can be performed; this yields a probability map for each class. To restore the original image dimensions, this map is upsampled by so-called deconvolutions. While pooling improves classification accuracy, it partially neglects spatial information. This is a drawback as it limits the spatial accuracy of the segmentation.

We can then use a so-called "skip" architecture [21] to solve this issue. Skip architecture - as the name suggests - skips some layer in the neural network and feeds the output of one layer as input to the next layer, as well as some other layer. What we want to achieve is to propagate information there was required for reconstruction during the up-sampling done using the FCN layer.

An example of a fully convolutional network is U-net [54]. One important modification in

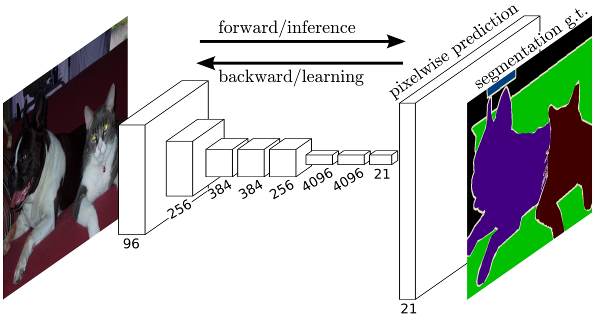


Figure 2.3: Shelhamer et al. in Fully Convolutional Networks for Semantic Segmentation.

U-Net to classical FCN is that there are many feature channels in the upsampling part, which allow the network to propagate context information to higher resolution layers. Consequently, the expansive path is more or less symmetric to the contracting party and yields a U-shaped architecture.

2.2.2 PSPNet

Pyramid Scene Parsing Network, a.k.a. PSPNet [76], is a semantic segmentation model that utilizes a pyramid parsing module to exploit global context information by different-region-based context aggregation. The PSPNet architecture takes into account the global context of the image to predict at the local level. This gives better performance on benchmark datasets like PASCAL VOC 2012 [18] and Cityscapes.

Like most semantic segmentation models, PSPNet is composed of two parts: an Encoder and a Decoder. The Encoder is characterized by a CNN backbone where the last traditional convolutional layers are replaced with Dilated convolution layers [74], a technique that expands the kernel by inserting empty spaces between its consecutive elements.

After the backbone we have the *Pyramid Pooling Module* (Figure 2.4), the major feature of PSPNet. This module pools feature map from the backbone to different sizes. Then it is passed through a convolution layer, after which upsampling takes place on the pooled features. This operation makes them the same size as the original feature map. Finally, the upsampled maps are concatenated with the original feature map to be passed to the decoder.

The PSPNet model is not a complete segmentation model in itself. It is just an encoder. It means it is just half of what is required for image segmentation. Usually, the most common decoders that are found in various implementations of PSPNet is a convolution layer followed by an 8x bilinear-upsampling [75]. However, this is not optimal for high-resolution output. In this context, it would be better to have a decoder that has learnable parameters and can take in intermediate features from the Encoder as input. To achieve this, we pass the feature map output through a feature pyramid network (FPN) decoder

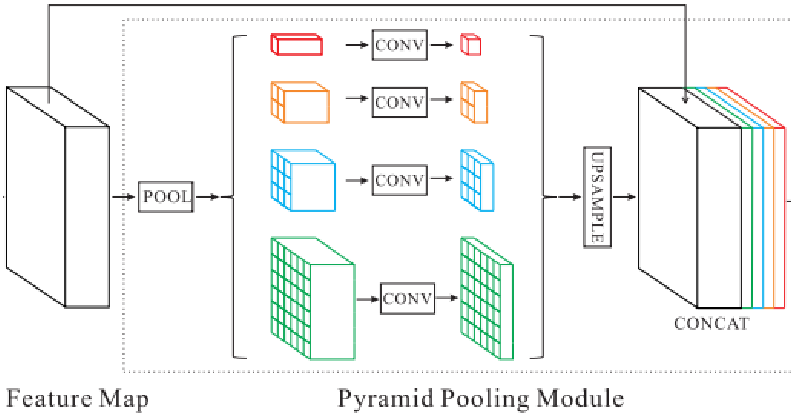


Figure 2.4: Pyramid Pooling Module overview.

[36], to improve the capabilities of PSP-Net. This way, we use an architecture close to U-Net to achieve an improvement on higher resolutions.

2.2.3 DeepLabV3+

DeepLab is a state-of-the-art semantic segmentation model designed and open-sourced by Google. Over the years, this model has evolved, and DeepLabV3+ [9] is the last version. Each innovative version brings some new ideas and significantly improves performance. The first presented version was DeepLab V1, which introduced the Atrous convolution concept. Unlike the other architectures, the feature maps are not downsampled by max-pooling. This component uses an algorithm called Atrous Convolution. Atrous convolution allows us to effectively enlarge the field of view of filters without increasing the number of parameters or the amount of computation.

DeepLab V2 instead introduced some new features in the field of semantic segmentation, e.g., Fully Connected Conditional Random Field (CRF), and Atrous Spatial Pyramid Pooling (ASPP) [8]. Atrous spatial pyramid pooling is an atrous version of spatial pyramid pooling, in which the concept has been used in PSPNet.

DeepLabv3 improves upon DeepLabv2 with several modifications. To handle the problem of segmenting objects at multiple scales, modules are designed which employ atrous convolution in cascade or in parallel to capture multi-scale contexts by adopting multiple atrous rates. Even ASSP module has been changed: global average pooling is applied, later it is fed to a 1 × 1 convolution and then bilinearly upsample the feature to the desired spatial dimension.

The last presented version is DeepLabv3+ extends DeepLabv3 by adding an encoder-decoder structure. The encoder module processes multi-scale contextual information

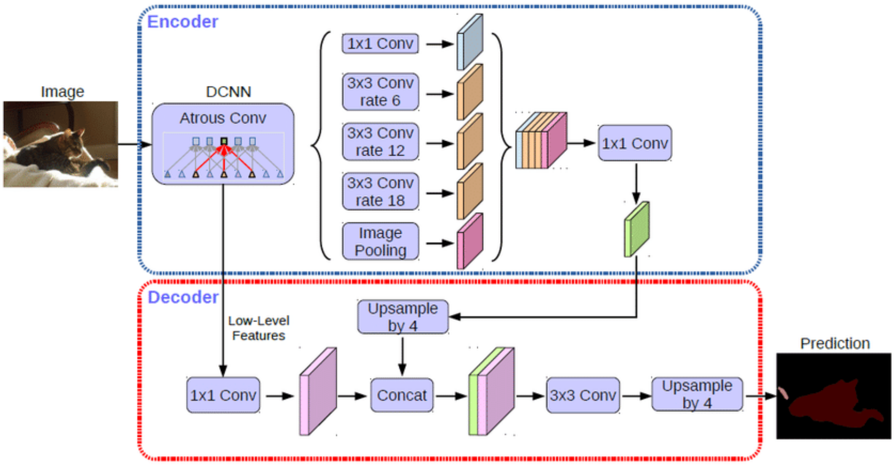


Figure 2.5: DeepLabv3+ architecture for segmentic segmentation.

by applying dilated convolution at multiple scales, while the decoder module refines the segmentation results along object boundaries. DeepLabV3 architecture is shown in Figure 2.5.

2.3 Generative adversarial networks (GANs)

Generative Adversarial Networks [22], or GANs, are a deep-learning-based generative model. Two models are trained simultaneously: a generative model G that captures the data distribution and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The idea is that the generative model is pitted against an adversary: the discriminative model learns to determine whether a sample is from the model distribution or the data distribution.
In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\mathcal{L}_{GAN} = \min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.1}$$

Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, log $(1 - D(G(z)))$ saturates. Rather than training G to minimize log $(1 - D(G(z)))$ we can train G to maximize log $D(G(z))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning. In Figure 2.6 a visual example of a GAN is shown.

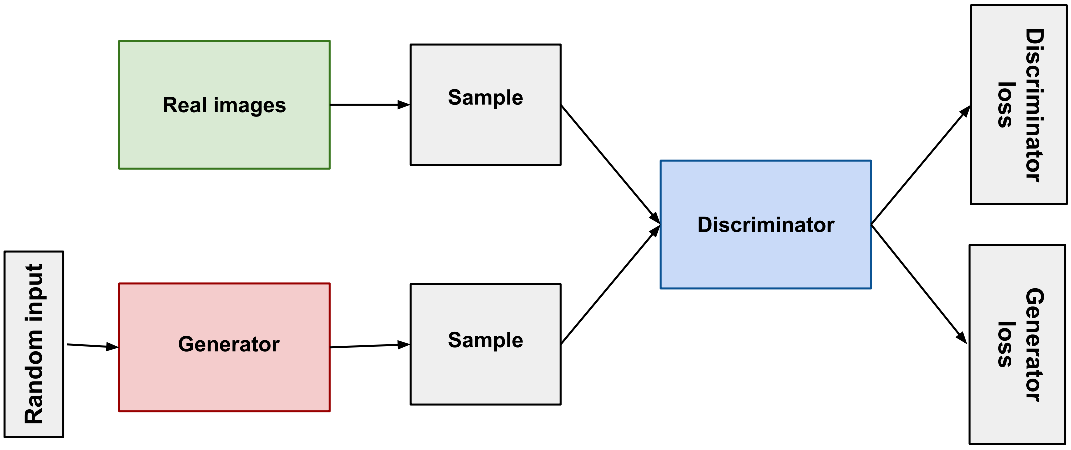


Figure 2.6: **Source**. Structure of Generative adversarial networks.

2.3.1 GAN Metrics

GANs use is spreading in machine learning community. Consequently, the need for robust metrics has been spreading as well. This being the reason that looking for metrics in this context has been a hot topic [71]. From the qualitative point of view, no objective loss function is used when training generative models, which suggests the evaluation should be done using the quality of the generated synthetic images. On the other side, from a quantitative point of view, when measuring how well our GAN performs, we need to evaluate two main properties:

- **Fidelity**, how realistic they are;
- **Diversity**, how much high the model output variety is.

With the aim to evaluate these two aspects we decided to use in this thesis **Fréchet Inception Distance** (section 2.3.2) and **Kernel inception distance** (section 2.3.3) as quantitative metrics. We used `clean fid` [47] library as implementation.
Furthermore, another good point for GAN evaluation is that synthetic data should be just as valuable as real data for the subsequent task when used for the same predictive purposes. Hence we decided to evaluate our images on the semantic segmentation task. We want that our synthetic images to help augment data in UAVs context.

2.3.2 Fréchet inception distance

The Fréchet inception distance (FID) [25] is a metric used to assess the quality of images created by a generative model. It is the current standard metric for assessing the quality

of generative models.
Rather than directly comparing images pixel by pixel, the FID compares the mean, and standard deviation of the deepest layer in Inception v3 [63] without its final classification layer. Specifically, the coding layer of the model (the last pooling layer prior to the output classification of images) is used to capture computer-vision-specific features of an input image. These activations are calculated for a collection of real and generated images. The activations are summarized as a multivariate Gaussian by calculating the mean and covariance of the images. These statistics are then calculated for the activations across the collection of real and generated images. At this point, we use Fréchet distance to calculate the distance between the distribution generated from real images and generated. The Fréchet distance formula is shown in Equation 2.2.

$$\text{FID} = ||\mu_r - \mu_g||^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}) \tag{2.2}$$

Lower FID is better, corresponding to more similar real and generated samples as measured by the distance between their activation distributions.

2.3.3 Kernel inception distance

FID usage has shown some limitations [10] over time: it is statistically biased, in the sense that on smaller dataset, the calculated value is not their true value, and it is quite slow. In this way, Kernel inception distance (KID) [5] has been proposed as a replacement for FID. It does not depend on the number of samples and is faster.
The general idea is close to the one presented on FID. It measures the dissimilarity between P_r and P_g for some fixed kernel function k. Given two sets of samples from P_r and P_g, the KID between two distributions can be computed with a finite sample approximation of the expectation. A lower KID means that P_g is closer to P_r. From a practical point of view, the KID equation is shown in Equation 2.3.

$$KID = MMD(I_{real}, I_{fake})^2 \tag{2.3}$$

where MMD is the maximum mean discrepancy and I_{real}, I_{fake} are extracted features from real and fake images. In particular, calculating the MMD requires the evaluation of a polynomial kernel function k (Equation 2.4), which controls the distance between two features. In practice, the MMD is calculated over a number of subsets to get both the mean and standard deviation of KID.

$$k(x, y) = (\gamma * x^T y + coef)^{degree} \tag{2.4}$$

2.3.4 CycleGAN

The Cycle Generative Adversarial Network [77], or *CycleGAN*, is an approach to training a deep convolutional neural network for image-to-image translation tasks. The idea is to learn how to translate an image from a source domain X to a target domain Y. Model learns a mapping $G : X \to Y$ such that the distribution of images from $G(X)$ is indistinguishable from the distribution Y using an adversarial loss. *CycleGAN* model can be viewed as training two autoencoders [3]: we learn one autoencoder $F \circ G : X \to X$ jointly with another $G \circ F : Y \to Y$. An example of the network structure is shown in Figure 2.7. To do that the characteristic of *CycleGAN* is the introduction of a cycle consistency loss to enforce $F(G(X)) \approx X$ (and vice versa). *Cycle consistency loss* is defined in Equation 2.5.

$$\mathcal{L}_{cyc}(G, F) = E_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + E_{y \sim p_Y(y)}[\|G(F(y)) - y\|_1] \tag{2.5}$$

This is not enough, so we have to use even the *adversarial loss* - used for training classical *GANs* - with the *cycle consistency loss.* Our full objective is shown in Equation 2.6.

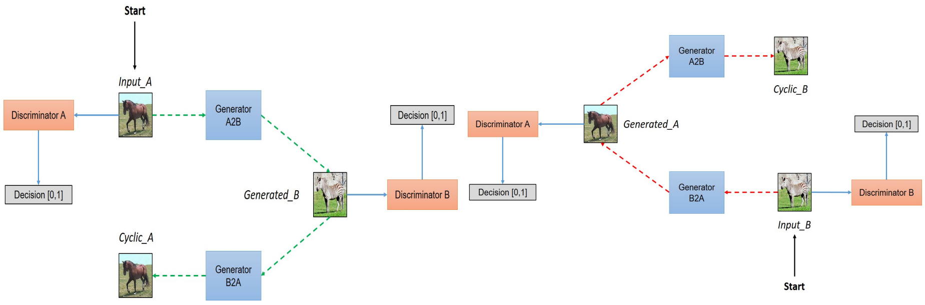
$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) \tag{2.6}$$

With λ that controls the relative importance of the two objectives. The final goal of the train is then to solve the Equation 2.7.

$$G^*, F^* = \min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y) \tag{2.7}$$

2.3.5 SG-GAN

Recent advances in vision tasks (e.g., segmentation) highly depend on the availability of large-scale real-world image annotations obtained by cumbersome human labor. *Semantic-aware Grad-GAN* (SG-GAN) [35] add to the classical structure of *CycleGAN* the ability to retain vital semantic information to virtual-to-real domain adaption. SG-GAN learns two symmetric mappings $G_{V \to R}$, $G_{R \to V}$ along with two corresponding semantic-aware discriminators SD_R, SD_V in an adversarial way. SG-GAN successfully



(a) $A \to B$ (b) $B \to A$

Figure 2.7: **Source**. Simplified view of CycleGAN architecture.

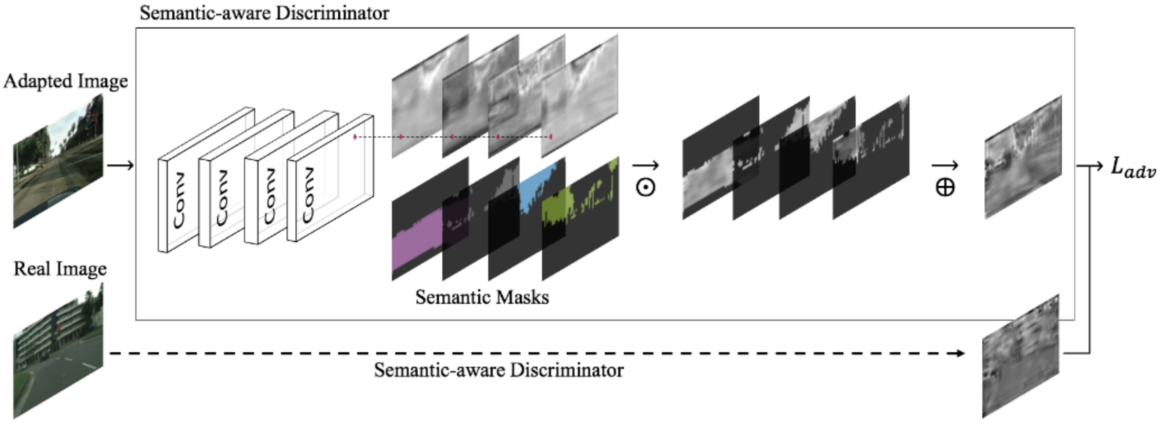


Figure 2.8: Illustration of a semantic-aware discriminator which takes either real or adapted images as inputs and is then optimized with an adversarial objective.

customizes the appearance adaption for each semantic region in order to preserve their key characteristic for better recognition. It presents two main contributions to traditional GANs:

- a soft gradient sensitive objective for keeping semantic boundaries;
- a semantic-aware discriminator for validating the fidelity of customized adaptions with respect to each semantic region.

To reach the first goal, the introduction of *soft gradient-sensitive loss* has been needed; it contributes to smoother textures and clearer semantic boundaries. On the network side, the discriminator is modified to allow for a better evaluation quality based on semantic masks. An image of the new discriminator is provided in Figure 2.8. SG-GAN successfully customizes the appearance adaption for each semantic region in order to preserve their the key characteristic for better recognition. This technique allows for an improvement in the final image generation with good accuracy on segmentation mask generation.

2.4 Synthetic-To-Real Domain Adaptation

Domain adaptation [69] is a field of computer vision where the goal is to train a neural network on a source dataset and get good accuracy on the target dataset, which is different from the source dataset. It has emerged as a new learning technique to address the lack of massive amounts of labeled data. DA can be split into two main categories based on a different domain divergence:

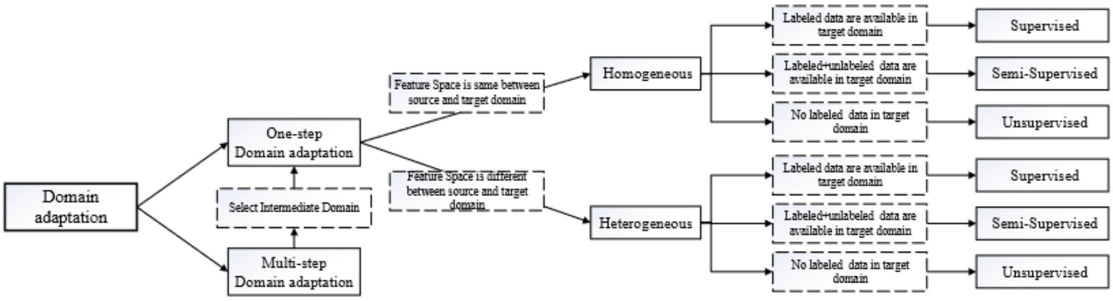

Domain adaptation
One-step Domain adaptation
Select Intermediate Domain
Multi-step Domain adaptation
Feature Space is same between source and target domain
Feature Space is different between source and target domain
Homogeneous
Heterogeneous
Labeled data are available in target domain
Labeled+unlabeled data are available in target domain
No labeled data in target domain
Supervised
Semi-Supervised
Unsupervised
Labeled data are available in target domain
Labeled+unlabeled data are available in target domain
No labeled data in target domain
Supervised
Semi-Supervised
Unsupervised

Figure 2.9: **Source.**An overview of different settings of domain adaptation.

- Homogeneous DA, source and target domains are identical regarding feature space, but they are generally different in terms of data distributions;
- Heterogeneous DA, the feature spaces between the source and target, domains are nonequivalent, and dimensions may also generally differ.

If we assume that the source and target domains are directly related we can call it *one-step DA*. In one-step DA, the deep approaches can be summarized into three cases:

- Class Criterion: uses the class label information as a guide for transferring knowledge between different domains;
- Statistic Criterion: aligns the statistical distribution shift between the source and target domains using some mechanisms;
- Architecture Criterion: aims at improving the ability of learning more transferable features by adjusting the architectures of deep networks.

It is not possible - most of the time - to find a direct correlation between source and target. In other words, there is a need for some intermediate representation to reach our goal. In these cases, we call it *multi-step (or transitive) DA*. In this thesis, we focus on Synthetic-To-Real translation [42], the domain adaptation task from synthetic (or virtual) data to real data. We are going to rely on an adversarial-based deep DA approach [19].

Chapter 3

Related Work

In this chapter, we are going to give an overview of various topics we have been investigating during this project development. A brief introduction to the current state-of-art for Synthetic-to-Real Domain Adaptation is given in section 3.1. Furthermore, in section 3.2 we are going to give a general overview of the state-of-art in the UAV machine learning community. The focus will be on Computer Vision and synthetic data tasks; we are going to ignore arguments such as autonomous UAVs drive. In conclusion, in section 3.3 and section 3.4 we are going to discuss some datasets we have to deal with during the development of this thesis.

3.1 Synthetic-to-Real state-of-art

Usage of synthetic datasets generated via simulators has already been widely explored in a large range of fields. In [62] a synthetic driving dataset is presented. The authors tested SHIFT dataset on a large amount of different computer vision tasks, results show how this approach helps model robustness and generality in autonomous driving systems. Regarding the task of synthetic driving dataset and urban segmentation context, there is already a framework able to generate datasets automatically. In [56], one Example of a framework to automatically generate datasets. Sekkat *et al.* presents a framework for generating omnidirectional images using images that are acquired from a virtual environment. In [52], GTA5 dataset - one of the most popular synthetic dataset - is presented. Richter *et al.* use *GTA V* as the environment to generate a large dataset containing 25 thousand images synthesized by a photorealistic open-world computer game. These images are provided with the corresponding semantic segmentation labels. In [55], another well-known dataset in the synthetic context is presented. The authors use SYNTHIA with publicly available real-world urban images with manually provided annotations. Then, they conduct experiments with DCNNs that show how the inclusion of SYNTHIA in the training stage significantly improves performance on the semantic segmentation

task.
In [27] current segmentation performance in Synthetic-to-Real from both SYNTHIA → Cityscapes and GTA5 → Cityscapes is provided. Authors obtain a multi-resolution training approach for UDA (Unsupervised domain adaptation). That approach combines the strengths of small high-resolution crops to preserve fine segmentation details and large low-resolution crops to capture long-range context dependencies.

3.2 A review on deep learning in UAV

Regarding UAVs, the use of GAN for data augmentation has already been researched. In [46] they used *CycleGAN* per data augmentation in the proposed DenseNet-based model. The experimental results showed how the proposed framework improves high wildfire detection accuracy. Another context where GAN models have been applied to UAVs is presented in [66]. The authors proposed a deep learning and generative adversarial network-based model for UAV low illumination image enhancement. LighterGAN was 6.66 percent higher than *CycleGAN* in subjective authenticity assessment and 3.84 lower in PIQE score. In the same direction, in [20], a novel method to generate a dataset for UAV software testing is proposed. The authors show that even if the dataset is disturbed, the images generated are still high quality.
Other really interesting techniques in UAVs context are discussed now. In [26], a Patch-Level augmentation technique is tested. Hong *et al.* generate multiscale chips to train object detectors. Next, these chips are used to construct an object pool and perform data augmentation. The resulting model is ranked 3rd in *VisDrone-DET2019 challenge.* On the other hand, in [14] *ROI* transformer is used for data augmentation. Authors use RoI Transformer to apply spatial transformations on RoIs and learn the transformation parameters under the supervision of oriented bounding box (OBB) annotations. This technique has achieved state-of-the-art performances on two common and challenging aerial datasets, i.e., DOTA and HRSC2016. In [61], a novel network - called the multiscale keypoint detection network (MKD-Net) - is proposed to detect multiscale objects in aerial scenes. MKD-Net fuses multiscale layers to generate multiple feature maps for objects of different sizes. Experiments on benchmarks PASCAL VOC and DOTA show impressive performance of MKD-Net compared with the baseline network. In [34], a Density-Map guided object detection Network (DMNet) for high-resolution aerial images is proposed by the authors. DMNet has three key components: a density map generation module, an image cropping module, and an object detector. DMNet achieves state-of-the-art performance on VisDrone and UAVDT. In [39] a new technique in UAVs preprocessing - called *Adaptive Resizing* - is introduced. The Adaptive Resizer is a preprocessing strategy designed to address bird's eye view (BEV) object detection, i.e., object detection from UAVs where the angle of view is pointing downwards at a right angle. Every image

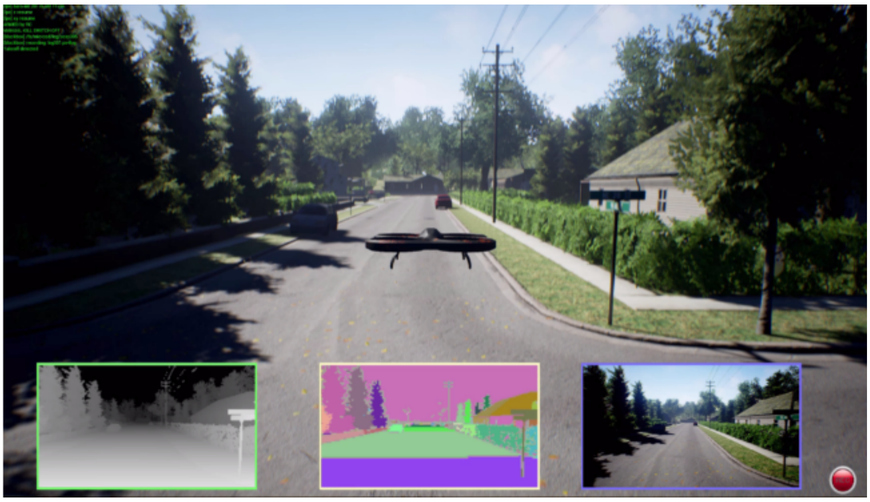


Figure 3.1: A snapshot from AirSim shows an urban environment in which a UAV is flying. The depth image stream, the materials property view stream and the front camera image stream are shown in real time in the inset.

is scaled in a principled manner to diminish the scale variance problem in BEV object detection. In [29], an approach based on Domain Bias in Object Detection on UAVs is shown. Authors demonstrate that domain knowledge is a valuable source of information and thus propose domain-aware object detectors by using freely accessible sensor data. This technique achieves a new state-of-the-art performance on UAVDT for embedded real-time detectors.

3.3 Synthetic datasets

Now we will discuss the current situation regarding UAV datasets and methods to acquire them. In section 3.3.1, we will give an overview of AirSim, a simulator made by Microsoft to generate a synthetic dataset on a large range of vehicles. Instead, in section 3.3.2 we are going to present *VALID*, a synthetic dataset acquired via AirSim. This dataset has been used in this thesis for experiment purposes.

3.3.1 AirSim

In 2017 Microsoft Research created Aerial Informatics and Robotics Platform - also called AirSim - [59] as a simulation platform for AI research and experimentation. AirSim is a simulator for drones and cars. It has been built on Unreal Engine and experimentally even on Unity. An example of AirSim running simulation is shown in Figure 3.1.
It is open-source, cross-platform, and supports software-in-the-loop simulation with pop-

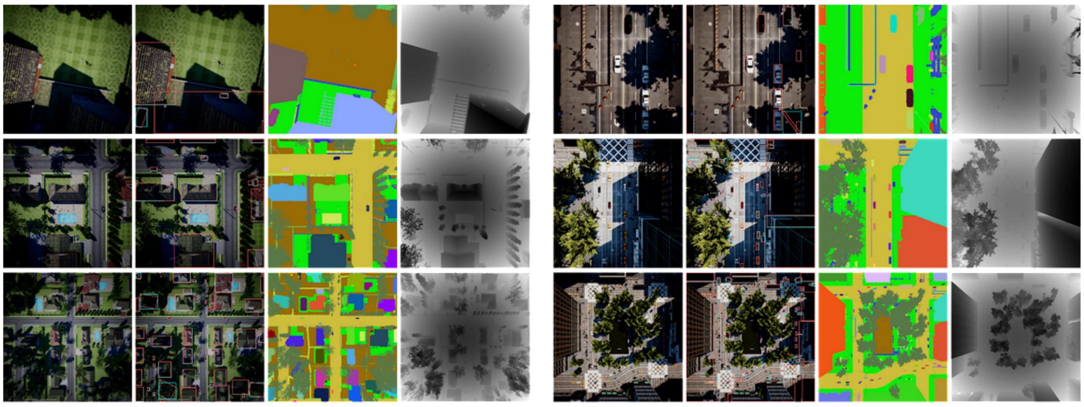


Figure 3.2: Example of data from VALID dataset.

ular flight controllers. It exposes APIs to retrieve data and control vehicles in a platform-independent way. This allows AirSim to support a wide range of platforms. The most significant advantage of this simulator is that it uses recent advances in computation and graphics to simulate physics and perception such that the environment realistically reflects the actual world.

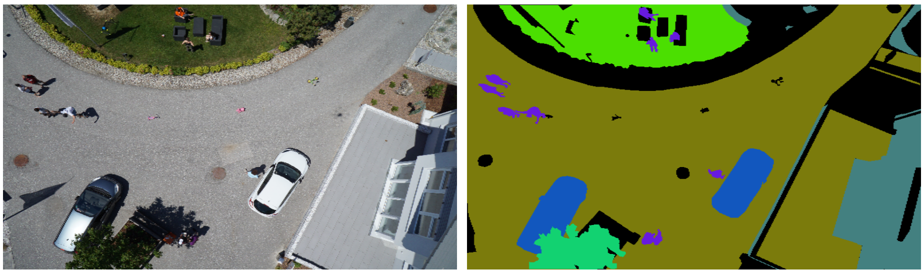
The simulator architecture [57] is described as follows: at the highest level, the simulator contains a model of the vehicle, the environment, and a physics engine to compute the resulting motions. In addition, AirSim allows support for recording sensor observations that mimic real-world behaviors. Finally, one of the significant components is photorealistic rendering via the Unreal engine, which enables computer vision analysis that is transferable to the real world. The framework also enables hardware-in-the-loop (HIL) and software-in-the-loop (SIL).

AirSim's goal is to develop a platform for AI research to experiment with deep learning, computer vision, and reinforcement learning algorithms for autonomous vehicles. We have not used it in this thesis because AirSim focuses on the vehicle's physics. We are more interested in easily making the label recording from the UAVs simulator parameterizable.

3.3.2 VALID

Virtual AeriaL Image Dataset, also called VALID [7], is a dataset coming from the implementation of a simulator that can simultaneously acquire diverse visual ground truth data in the virtual environment. AirSim has been used to deploy it.

VALID consists of 6690 high-resolution images (1024×1024), all annotated with panoptic segmentation on 30 categories, object detection with the oriented bounding box, and binocular depth maps. They have been collected in 6 different virtual scenes and five various ambient conditions (sunny, dusk, night, snow, and fog). It is presented as the



(a) Image

(b) Label

Figure 3.3: Example of data from Semantic Drone Dataset.

first dataset specifically designed for aerial scenes. An example of images from VALID is shown in Figure 3.2.

Multiple scenarios have been considered to mimic the real world: crowded airports, downtown with skyscrapers, the neighborhood with villas, European-style streets, hilly seaside towns, and primitive snow mountains. Images have been taken at three altitudes: 20m, 50m, and 100m.

On the annotation side, VALID consists of 20 main categories, i.e., tree, plant, road, pavement, land, water, ice, rock, bridge, sign, vehicle, building, animal, person, common obstacle, high obstacle, tunnel, ship, plane, and harbor. These categories are then refined into several categories, e.g., water is further divided into natural water and swimming pools. In the end, 30 detailed categories are formed. To achieve panoptic segmentation, 17 categories are recognized as ”thing”, and the rest 13 are classified as ”stuff”. Instance segmentation and object detection tasks are based on ”thing” categories, while semantic and panoptic segmentation tasks use all 30 detailed categories.

3.4 Real datasets

In this section, we are going to discuss real UAV datasets that have been used during this project. They have been needed for the transfer learning part and as a benchmark in the evaluation part. On the transfer learning side we have mainly used Semantic Drone Dataset (section 3.4.1), FloodNet dataset (section 3.4.2) and LoveDA (section 3.4.3). LoveDA images have been acquired from satellite, so they are not UAVs taken images. However, we have been using them because the two fields are really close to each other. On the benchmark side, we have used UAVid, discussed in section 3.4.4.

3.4.1 Semantic Drone Dataset

The Semantic Drone Dataset [43] focuses on the semantic understanding of urban scenes. The imagery depicts more than 20 houses from a nadir (bird's eye) view acquired at an altitude of 5 to 30 meters above the ground. A high-resolution camera was used to acquire images at a size of 6000 × 4000 px. The training set contains 400 publicly available images, and the test set is made up of 200 images. An example is shown in Figure 3.3.

A pixel-accurate annotation for the same training and test set has been provided. The annotated classes are 20. Annotations range from people to paved areas. They have also annotated natural objects (such as trees, grass, other vegetation, dirt, gravel, and rocks) and house objects, e.g., wall, window, fence, door, roof. Water and pool have been divided. Furthermore, vehicles (car and bicycle) have been annotated too.

3.4.2 FloodNet Dataset

FloodNet [49], is a dataset composed of images captured after hurricane Harvey. Hurricane Harvey made landfall near Texas and Louisiana in August, 2017, as a Category 4 hurricane. The images are labeled pixel-wise for semantic segmentation task and questions - for question and aswering model - are produced for the task of visual question answering. FloodNet poses several challenges including detection of flooded roads and buildings and distinguishing between natural water and flooded water.

As we are primarily interested in semantic segmentation, we will ignore the Visual Question Answering. The pixel-wise annotation they have done on the images is essential

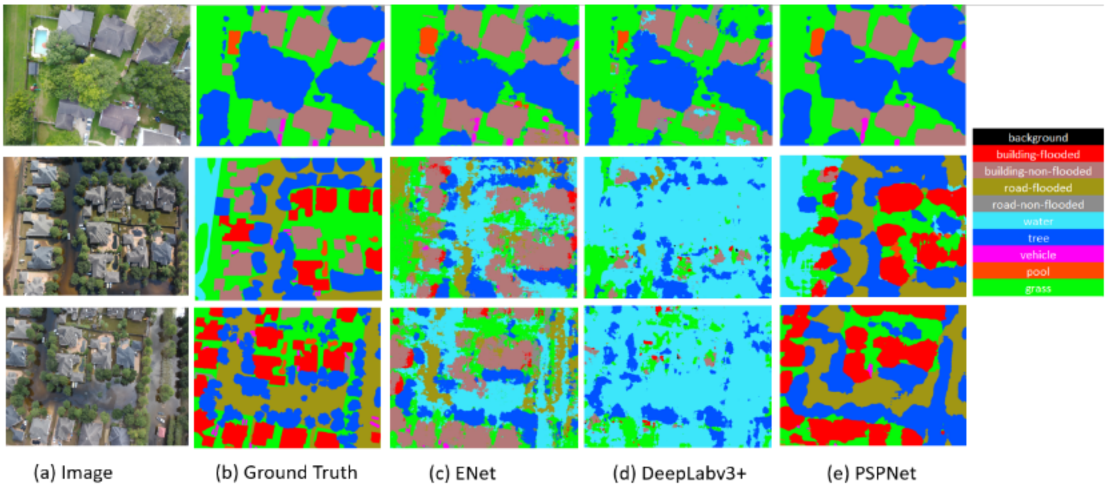


Figure 3.4: Visual comparison on FloodNet test set for Semantic Segmentation.

for this thesis. In total 3200 images have been annotated with 9 classes which include building-flooded, building-non-flooded, road-flooded, road-non-flooded, water, tree, vehicle, pool, and grass.
The authors tested the dataset on a semantic segmentation task using three nets: ENet [48], PSPNet and DeepLabV3+. As evaluation metric they used mean IoU (mIoU). This test shows us how PSPNet is clearly able to out-perform both the other networks. PSPNet is giving the best score (79.69%) on basically each category, and usually it is able to almost double perform *ENet*. It is interesting to note that although DeepLabv3+ and PSPNet collect global contextual information; their performance in detecting flooded building and flooded roads are still low since distinguishing between flooded and non-flooded objects heavily depend on the respective contexts of the classes. This work gave us - as well as an excellent level of annotation data - a direction on which networks could fit our tasks and which did not.

3.4.3 LoveDA

The LoveDA [67] dataset contains 5.987 HSR (High Spatial Resolution) images with 166.768 annotated objects from three cities: Nanjing, Changzhou, and Wuhan. The focus has been posed on different geographical environments between Urban and Rural. The historical images were obtained from the Google Earth platform. As each research area has its planning strategy, the urban-rural ratio is inconsistent.
There are nine urban areas selected from different economically developed districts, which

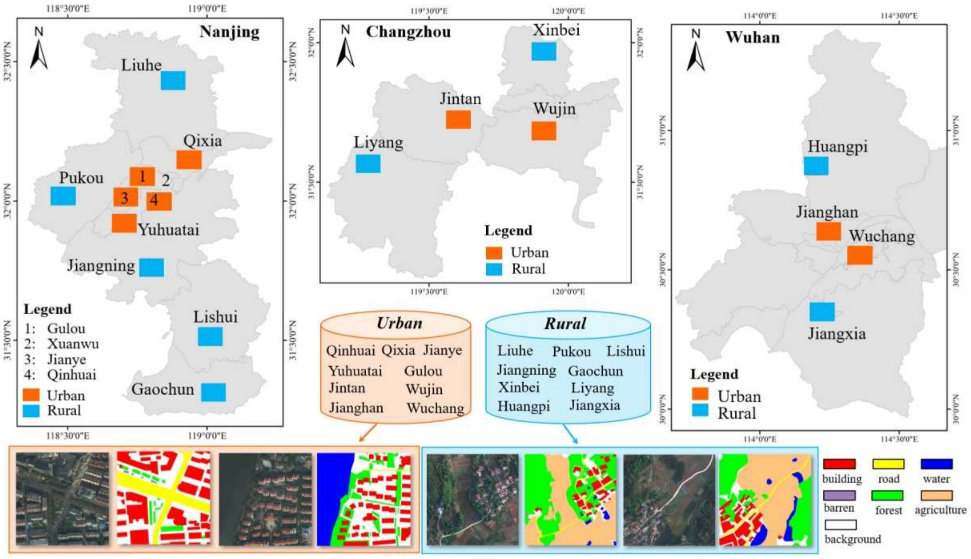


Figure 3.5: Overview of the dataset distribution. The images were collected from Nanjing, Changzhou and Wuhan cities.

are all densely populated ($> 1000\ people/km^2$). The other nine rural areas were selected from undeveloped districts. After geometric registration and preprocessing, each area is covered by 1024 × 1024 images, without overlap. The joined images cover 536.15 km^2. Annotated data on semantic segmentation are buildings, roads, water, forest, barren, forest and agriculture.

3.4.4 UAVid

UAVid [38] is a semantic segmentation dataset for UAV Imagery. It proposes a new challenge: semantic segmentation on the UAV dataset with moving object recognition and temporal consistency preservation. Furthermore, the proposed dataset works in a 4K resolution. This increases the already high level of the challenge.

The UAV dataset consists of 30 video sequences capturing 4K high-resolution images in slanted views. In total, 300 images have been densely labeled with 8 classes for the semantic labeling task. As far as we know, this is one of the most densely annotated UAV dataset. The classes that have been annotated are: building, tree, clutter, road, vegetation, static car, moving car and human.

The structure of the annotation introduces some challenges itself. Annotations are unbalanced: building and tree have more annotations than the rest of the classes. Instead, static and moving cars - same as humans - are missing annotations. Moreover, it's im-

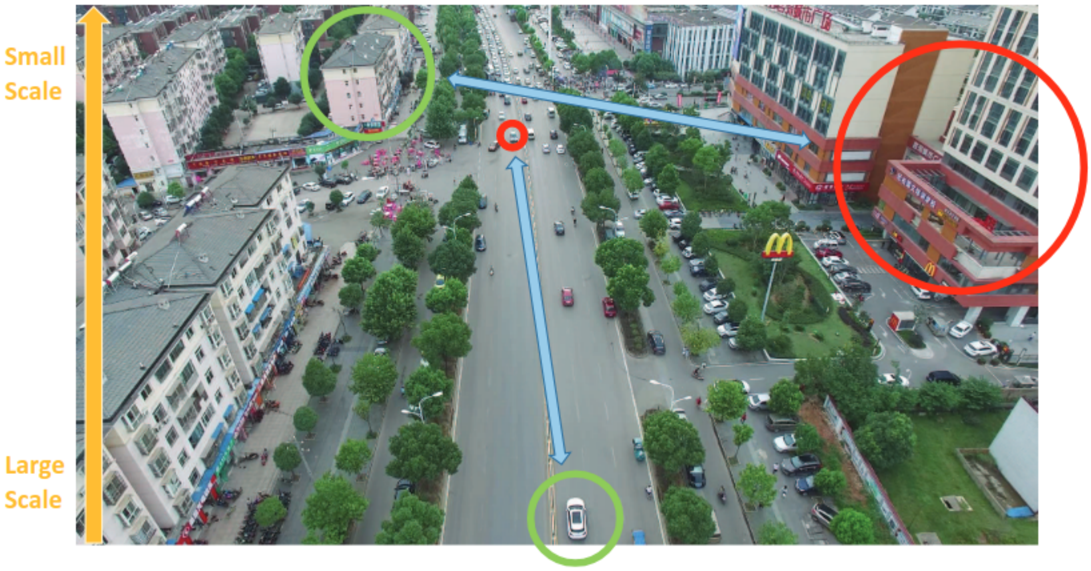


Figure 3.6: Illustration of the scale problem in a UAV image. The green circles mark the objects in proper scales, while the red circles mark the objects in either too large or too small scales.

portant to mention that the clutter class has a relatively large pixel number ratio and consists of meaningful objects, which is taken as one class for both training and evaluation rather than being ignored.
Using such a larger resolution in a UAV context introduce some interesting side challenges, for Example the scales of the objects vary greatly from the bottom to the top of the image. This means the network we train should not just learn from one prospective and from a defined depth: we need to learn the same object at different scale. This particularity is given by the particular prospective that is used: oblique images from UAVs. What we discussed here is better illustrated on Figure 3.6.
In conclusion, in the presentation paper, authors have shown some machine learning networks implementation to be trained on this task. The best results they get is a value of 50.1% on mean IoU (mIoU) using a pre-trained MS-Dilation network (even this network has been presented on the paper). Given the described challenges and the given results we decided to use this dataset as benchmark for our work. We are interest on understand if we can over-perform the original results and if synthetic data can help us to do that.

Chapter 4

Simulator

In this chapter, we present our simulator. A *simulator* is a software that artificially creates the effect of being in a certain environment. In our case, we developed a flight simulator, a device that artificially re-creates aircraft flight and the environment in which it flies. We discuss in section 4.1 what *Unreal Engine* is. We give a brief introduction to its history and how it is born. Section 4.2 is used to introduce the ideas behind the simulator's architecture. In section 4.3 we introduce how the server works and what it is needed for. The server is the kernel of the simulator: it receives the request from the client (section 4.5) - via Python library API (section 4.4) - and process them to generate semantic segmentation mask, instance segmentation mask, bounding boxes and depth mask.

4.1 Unreal Engine

Unreal Engine [17], or UE5 in its current version, is an open-source, real-time 3D game engine created initially by Epic Games in 1998.

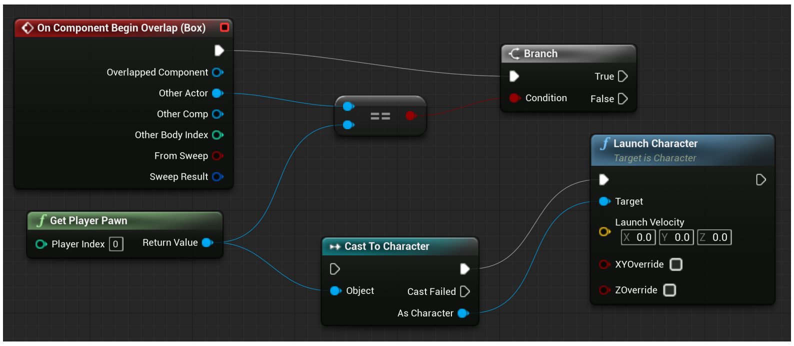


Figure 4.1: An example of a `Blueprint`.

The main goal of a game engine [2] is to abstract standard video game features, allowing for code and game asset reuse in different games. Some of those features are input handling, game loop, and memory management. The most recent game engine - and that of UE - went further and can even simulate physical laws such as gravity and collisions, as well as game components like audio, video, artificial intelligence, virtual reality, and many more features. Given the described characteristics, engine usage spread over the programmers' community: they easily allow code reuse and give the programmer the possibility to have thousands of tools just out of the box.

Even though UE was developed for gaming purposes, its use has spread in the simulator field. There are many more reasons why using Unreal Engine as an engineering simulator is an advantage: usage of the graphical and physics motors and real-time simulation relieves the user from working on non-essential issues such as collision detection, physical laws, graphics, etc.

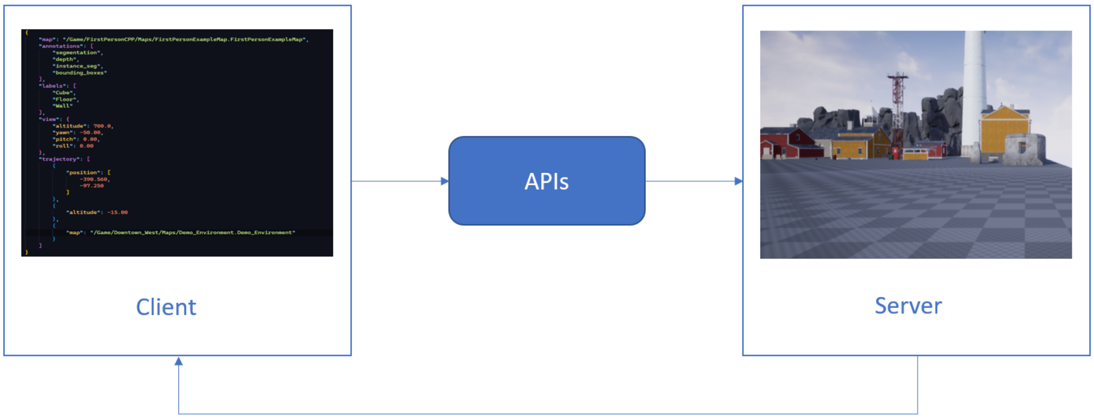
Furthermore, UE allows the programmer to focus on defining at a higher level the idea he has. The focus is on programming the behaviors that the objects in the world - such as vehicles - have in relation to it. In fact, UE allows the programming to use two different programming languages: `C++` and `Blueprint`. `C++` does not need to be presented. Instead, `Blueprint` is a programming language that respects previously described philosophy. It consists of a node-based interface to create gameplay elements from within the editor. Then, it moves the abstraction to another level. The programmer is not focused on writing code but on defining a `Blueprint` for the object via graphical nodes. A `Blueprint` example is shown in Figure 4.1.

As a downside, Unreal Engine may feel harder to learn and slower to prototype than other software. The variety of tools is wide [1], so it is not easy to master the entire pipeline. Despite that, the availability of already created and programmed assets (such as the one used in this thesis) and plugins provides a quick way to integrate them and allows the programmer to ignore some parts of the toolchain.

In conclusion, given the trade-offs previously described, we decided to use UE as the starting point for this thesis's development. Although we want to focus on a more abstract level, our goal is to build a pipeline that provides us with well-annotated data in a parametric way. We can discard topics such as the physical system used on the simulator or the audio system. For these reasons, UE fits us.

4.2 Architecture

We are not interested in building monolithic software; we would prefer to build a simulator that can be easily adapted to the programmers' needs. For this reason, we adopted a server-client architecture for the simulator. In this way, we could decouple as much as we can the tasks. We mainly focused on the server and the communication system between server and client.



```
{
    "map": "/Game/FirstPersonCPP/Maps/FirstPersonExampleMap.FirstPersonExampleMap",
    "annotations": [
        "segmentation",
        "depth",
        "instance_seg",
        "bounding_boxes"
    ],
    "labels": [
        "Cube",
        "Floor",
        "Wall"
    ],
    "view": {
        "altitude": 700.0,
        "yawn": -50.00,
        "pitch": 0.00,
        "roll": 0.00
    },
    "trajectory": [
        {
            "position": [
                -390.560,
                -97.250
            ]
        },
        {
            "altitude": -15.00
        },
        {
            "map": "/Game/Downtown_West/Maps/Demo_Environment.Demo_Environment"
        }
    ]
}
```

Figure 4.2: An overview on the simulator architecture.

The server is responsible for the image rendering process and the map and object handling. Furthermore, it generates the segmentation, depth, and instance masks. Another task the server is responsable for is exposing the APIs to communicate with it. We provide our system as a UE plugin, in this way, it can be installed on every other UE project leaving the same functionalities.
Handling API calls can generate some difficulties and performing some action can turn out into a repetitive task. For this reason, we provide a Python Library `UAVS2R.py` to handle the communications between server and client easily. We want the user to be transparent about what is happening under the hood. The programmer should focus on what kind of parameters the simulator should be instantiated at that moment.
To show the functionality of our software, we built a JSON parsing client that wraps `UAVS2R.py`. The user can feed the client a well-formatted JSON file that can give pieces of information over the simulation status to the server. Detaching `UAVS2R.py` to the client allows the programmer to build a custom client with whatever mechanism he wants.

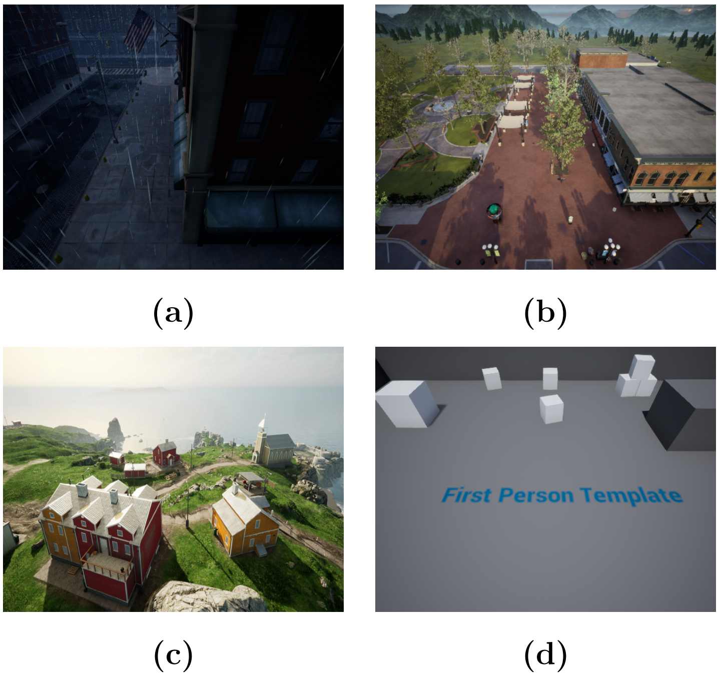
4.3 Server

In this paragraph, we are going to discuss how the server works. The purpose of the server is to provide out-of-the-box tools to generate segmentation, instance, and depth mask from rendered scenes in various environments. In order to ease the usability for users, we provide various environments to work with. They are:

- A urban environment that closely mimics a metropolitan context, a preview is visible in Figure 4.3a;

- An environment that tries to mimic a mixed park and urban context, a preview is visible in Figure 4.3b;
- An environment based on Nordic island panoramas, Figure 4.3c;
- A debugging environment, the default one, Figure 4.3d.

The urban environments are provided with different weather conditions. The user can even specify various metadata to each image the simulator takes.
We fork UnrealCV [70] to reach our purposes. This UE plugin allowed us to simulate semantic/instance segmentation and place and move objects in the scene. However, what we want at the end is something closer to CARLA Simulator [15]. UnrealCV is quite limited on the communication part. It allows changing segmentation color at run-time, for example. Nevertheless, it does not allow us to make it in a structured way. The user can change color object by object. Instead, we want something like CARLA, where the user can easily assign the same label to multiple classes. We reached this goal by allowing the programmer to use a mechanism of label assignment based on a criterion. We will discuss this deeper in section 4.4.
This plugin was only available on Unreal Engine 4.27, so we worked to implement it



(a) (b) (c) (d)

Figure 4.3: An overview on the provided environments: **(a)** a metropolitan raining environment, **(b)** a urban environment, **(c)** a nordic island environment, **(d)** a debugging environment.

even on UE5. The engine switch has shown some difficulties: we had to modify multiple classes from the original implementation. Over time, Unreal Engine has changed a lot, and various classes have been deprecated in the last release (*UE5*). Then part of the work has been to run on this adaptation of the source code.
We have even tried to go further than the original implementation: we added the possibility to switch to a different environment at run-time and add a bounding-boxes generation system. The map switch is particularly convenient. Given that we want to let our simulator run parametrically we can switch the map we are using at run-time, allowing us to speed up the image acquisition process.
The basic behavior of the simulator is to always wait for the next operation. The server is waiting for a client message that says what it should do. Operations could be of two types:

- `vset`, used to modify objects and information in the scene;
- `vget` used to retrieve informations about objects in the scene.

Below is shown the structure of `vset` command:

```
vset /{object_type}/{id}/{operation} (args1) ... (argsN)
```

Where the `{object_type}` is the type of object you want the operation is performed on and `{id}` is the unique identifier of the object.
Then if we want to perform a camera rotation, the command we should send to the server is:

```
vset /camera/0/rotation 20.0 30.0 7.0
```

Where the three arguments are respectively: *yawn*, *pitch* and *roll* values. Usually, `vget` is specular to `vset` but without arguments. Then, to retrieve the camera rotation what we have to do is:

```
vget /camera/0/rotation
```

It is possible even load objects at run-time. This is important because it allow the user to update the scene dynamically during the annotation process. An example of how add an object to the level is:

```
vset /objects/spawn UClass_name
```

Worth to be mentioned how to switch the map. It can be easily done using the command:

```
vset /level {map}
```

where the `{map}` value is registered in *Unreal Engine Assets Registry.* So the user can even use `Pak` (the assets handler in UE) to load a new map at run-time and then switch to it. No method to retrieve which map the user is using has been implemented.

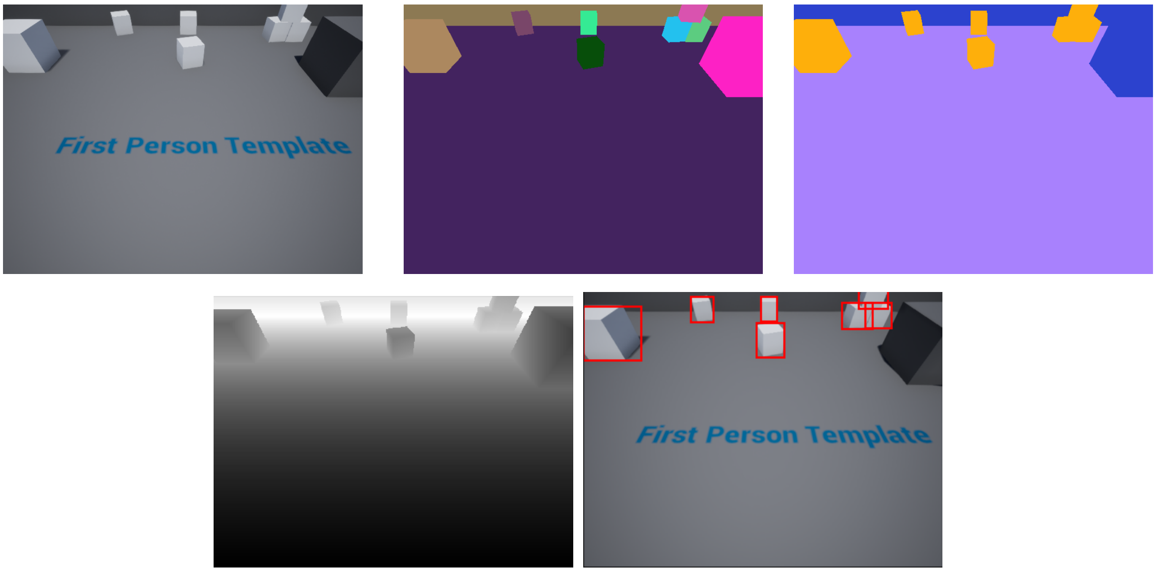


Figure 4.4: Simulator output example. In the first row - from left to right - are shown: original image, instance segmentation, semantic segmentation. In the second row depth mask and bounding boxes are shown.

4.4 APIs

So far, we have discussed the structure of the server. However, we now need to discuss a way to communicate with it. As said before, we tried to decouple the interactions between our server and the client as much as possible. We build a standalone Python library to reach this purpose. We called this library `UAVS2R.py`.
To start the communication with the server, the server should instantiate an `DefaultMap` object. The object constructor takes as input a list of labels and the map's name (optional). Labels are defined by the user. The object instantiation will perform the connection with the server. By default, a color will be assigned to each label defined in the constructor. From now the user can interact with the server. The idea is that we transpose the API call that the server exposes to methods of this class. This means that the following command:

```
vset /camera/0/rotation 20.0  30.0  7.0
```

is simply translated in our *Python* library as shown in Figure 4.5 on line 7. However, the user can even add some metadata to the annotation. On line 9 of Figure 4.5 we are saying to the server to annotate that weather is rainy, and the wind speed is $\approx 20km/h$. The annotation will be propagated to all the next images in this environment, even though it can always be overwritten. Bounding boxes are generated on the client side via the library. Using a criterion - defined via `get_bb_criterion()` - the user is

```
1  map = DefaultMap([
2      "Cube",
3      "Wall",
4      "Floor"
5  ])
6
7  map.set_rotation([-20, 30, 7])
8
9  map.set_meta({
10     "weather": "rainy",
11     "wind_speed": "almost 20km/h"
12 })
13
14 map.save("dataset/",[
15         "segmentation",
16         "depth"
17     ])
```

Figure 4.5: Code snippet using the *Python* library.

able to acquire the bounding boxes from the defined labels. The default behavior is to acquire the bounding boxes based on instance segmentation labels. Afterwards, we performed the operation we want, we can now generate annotations. We can call the method `save(path, annotations)` to generate an annotation, as shown on line 14 in Figure 4.5. This method will create in the required `path` some folders - one for each required annotation - where the annotation will be stored. For example - the line that is shown in the code snippet - will generate inside the folder `dataset/` two folders: `segmentation/` and `depth/`. Where the semantic segmentation mask and the depth mask are stored. Furthermore, a folder called `annotations/` is created. Inside that folder, the user will find all the general information about the environment, e.g., label color, metadata, etc.

Semantic and instance segmentation mask are stored as `.png` file, instead, depth mask is annotated as `.npy` file. The file containing the general annotations is a `.json` file.

The out-of-the-box labeling system we thought is pretty straightforward. Using a *regex* all the objects are queried, and if a match is found, we assign the given label to that specific object. In the same way, the naming file system is pretty simple, an id is generated for each annotation and each image/annotation is stored with that id as name.

As we said before, we tried to generalize the server-client interaction as much as possible. For this reason, we defined our communication system as a *Python* class. In this way, via inheritance, the programmer can always extend our `DefaultMap` class to let it behave

```
class AdvancedMap(DefaultMap):
    ...
    def associate_label_object(self, labels):
        objects = self.get_objects()

        list_objects = {object: [0, 0, 0] for object in objects}

        for object in objects:
            for label, color in labels.items():
                if len(label) > 1:
                    label = label[0]
                if label in object:
                    list_objects[object] = color

        return list_objects
    ...
    def get_filename(self):
        return "date_" + str(uuid.uuid4())
    ...
   def get_bb_criterion(self):
        return self.get_semantic_mask()
```

Figure 4.6: Example of `DefaultMap` class inheritance.

as he wants. With this idea in mind, we defined some methods for defining criteria over annotation operations. This is the case of `associate_label_object(labels)` that is responsible to associate at each object a label. The user can always redefine the behavior to match object labels, not via regex but another criterion. The same point is valid for `get_bb_criterion()`, where you can define a criterion for bounding boxes generation. It's not always said that the bounding boxes generation should exactly match the instance mask (default behavior). This system offers the programmer a higher level of freedom: he can customize the annotation generation as he wants. An example of class inheritance is shown in Figure 4.6 an example of generated annotations is shown in Figure 4.4.

4.5 Client

We were not interested in creating just one way to communicate with our server. As we said, our interest is to build a modular parameterized framework. So the user should not think this client is the only one possible. The platform is thought that its terminal

```json
{
    "map": "FirstPersonExampleMap",
    "annotations": [
        "segmentation",
        "bounding_boxes"
    ],
    "labels": [
        "Circle",
        "Cube",
        "Wall"
    ],
    "view": {
        "altitude": 3000.0,
        "yawn": -35.00,
        "pitch": 0.00,
        "roll": 0.00
    },
    "trajectory": [
        {
            "position": [
                -370.560,
                -97.250
            ]
        },
        {
            "altitude": 900.00
        },
        {
            "rotation": [
                [
                    -40.0,
                    0.0,
                    0.0
                ]
            ]
        }
    ]
}
```

Figure 4.7: An input JSON file example.

part could permanently be removed and replaced with one that better fits user purposes. What we provide, then, is closer to proof-of-work.

The implementation we will show now is based on a JSON file. This JSON file should contain a JSON object with five attribute: `map`, `annotations`, `labels`, `view` and `trajectory`. The `map` attribute is used to define the starting map. Instead, `annotations` is an attribute used to define which kind of annotations we are looking for. This attribute is defined as a list, the values that can be passed are: *segmentation*, *depth*, *instance_seg* and *bounding_boxes*. The `labels` define which labels are the user interested in annotating. They can be passed as a string, and a match between the object name and passed labels is done. The last two attribute are `view` and `trajectory`. The behaviour of `view` is straightforward: it takes an object where *yawn*, *pitch*, *roll* and *altitude* are defined. They are the starting position, you can always move them. The kernel of this implementation is the attribute `trajectory`: it defines the movements the UAV should do. It is a list of JSON object, and each object could be seen as a command. The command type list is: *position*, *rotation*, *altitude*, *level* and *metadata*. A fully example on how an input file is composed is shown in Figure 4.7.

As we said, this implementation is quite naïve. However, it was enough for our purposes and as proof-of-work. A possible future implementation is in ROS [60], an open-source framework that helps researchers and developers build and reuse code between robotics applications. This could allow our simulator to cooperate with other software to improve the annotation quality and acquisition.

Chapter 5

Domain Adaptation

During the development of this thesis, we have been testing multiple solutions for the part of photo-realism augmentation. In this chapter, our primary focus has been on *CycleGAN*. We will discuss our experiments with this network in section 5.1. Although *CycleGAN* is not semantically guided: this is the reason we have tested *SG-GAN*, a semantic guided network inspired by CycleGAN. We will discuss this network's results in section 5.2. In addition, several other methods have been investigated to generate synthetic images [64]. Nevertheless, we restricted this thesis to explore some preliminary feedback, and the discussed networks were the ones most fitting our aims. These are the reasons we are not going further. In the future, our focus could be given to networks precisely thought for Synthetic to Real Domain Adaptation, as [65].

5.1 CycleGAN

So far, we have a simulator that can produce images with high-quality annotations. However, images given from the simulator still suffer from classical problems of images taken via game engines: when the quality of meshes is not too high, the look and feel are always a little "plastic".
Obviously, good quality meshes require considerable effort to be collected. For this reason, we decide to use *CycleGAN*. In this way, we should be able to augment realism without particular effort. This is not an easy task. UAVs are not all equals: the camera mounted on a UAV is not from just one point of view.
For this reason, we had to test a couple of sources and target dataset combinations. We have been discussing a couple of datasets in the background (chapter 2).
These datasets have been used for more than one run on *CycleGAN* with different combinations to evaluate the best possibilities this network can offer us on this task. Furthermore, we tested these datasets on different resolutions.
Our first test starts with an image resolution of 256×256. This has been done to reduce

the amount of information needed on the first steps of this work. Initially, we selected two environments from the VALID dataset: urban and metropolitan. These images were used as source datasets; instead, images from VisDrone were used as the target domain. This experiment shows us that the path we were pursuing was promising. Despite this, these results were not enough. Most of the time a strange glitch effect was present. Some shrouds were visible in the final images. Furthermore, the human feedback we had was that they looked more natural, but they were not looking realistic enough. From one side, a lighting improvement was visible. On the other side, shroud and prospective error were omnipresent.

Firstly we focused ourselves on resolving these prospective errors. We decided to shift our translation work to a perpendicular point of view. We took this strong decision because images from VALID are taken from that point of view. This way, all the objects are visible from the same point of view in the source and target domain. Finding a dataset with a valid data amount for the training was difficult.

Usually, UAV datasets are not big enough for this kind of data and are not from a coherent point of view. For this reason, we needed to join multiple datasets into one. It is now that REAL was born. REAL dataset is how we called the combination of *Semantic Drone Dataset*, *FloodNet* and *LoveDA*.

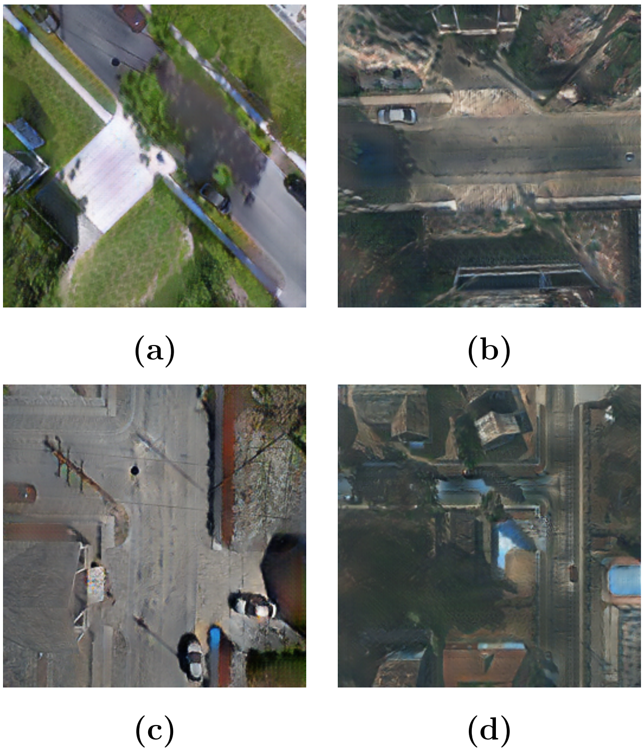
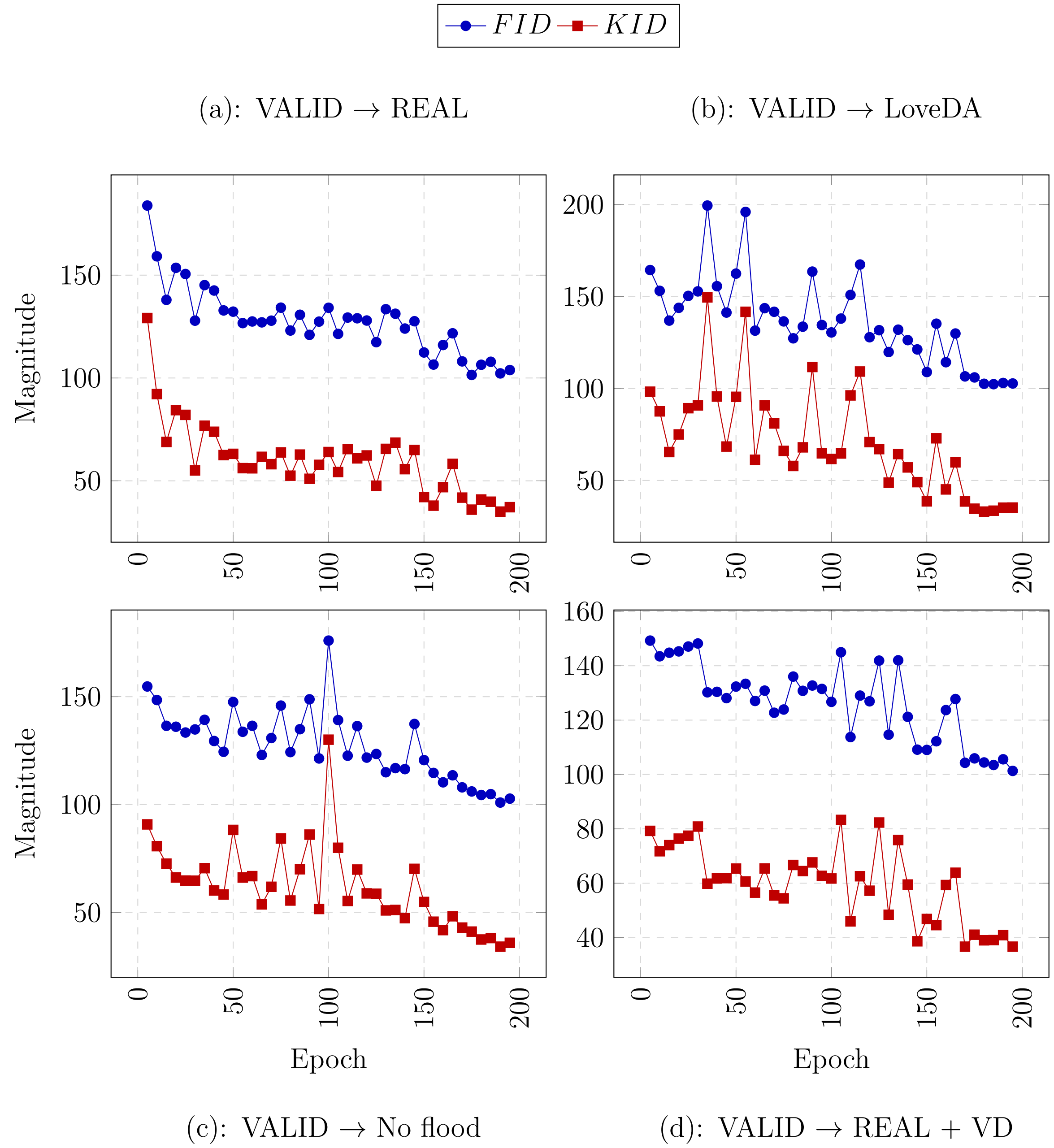


Figure 5.1: Some examples of images generated via *CycleGAN* at resolution of 256 × 256.



(a): VALID → REAL

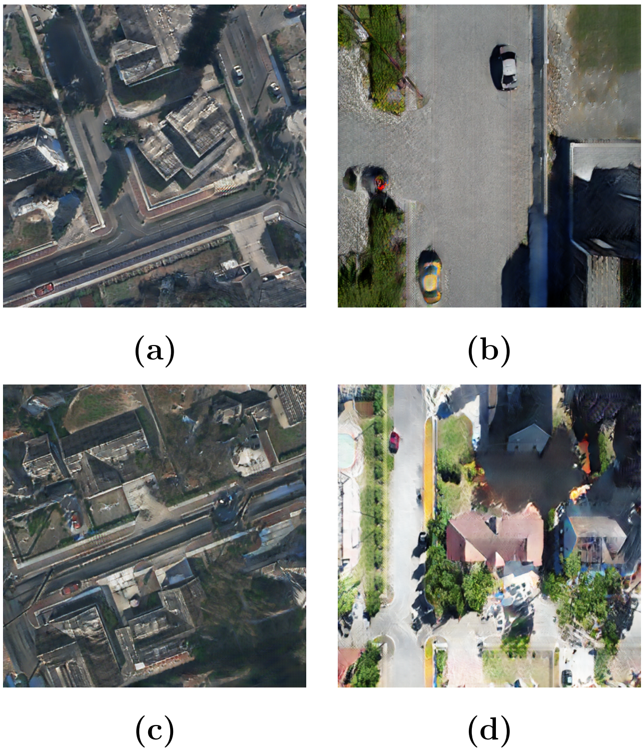
(b): VALID → LoveDA

(c): VALID → No flood

(d): VALID → REAL + VD

Figure 5.2: *Fréchet Inception Distance* and *Kernel Inception Distance* during 200 epochs of training. Image source domain X is fixed but target domain Y changes on each plot. Images resolution is 256 × 256. A lower score is better.

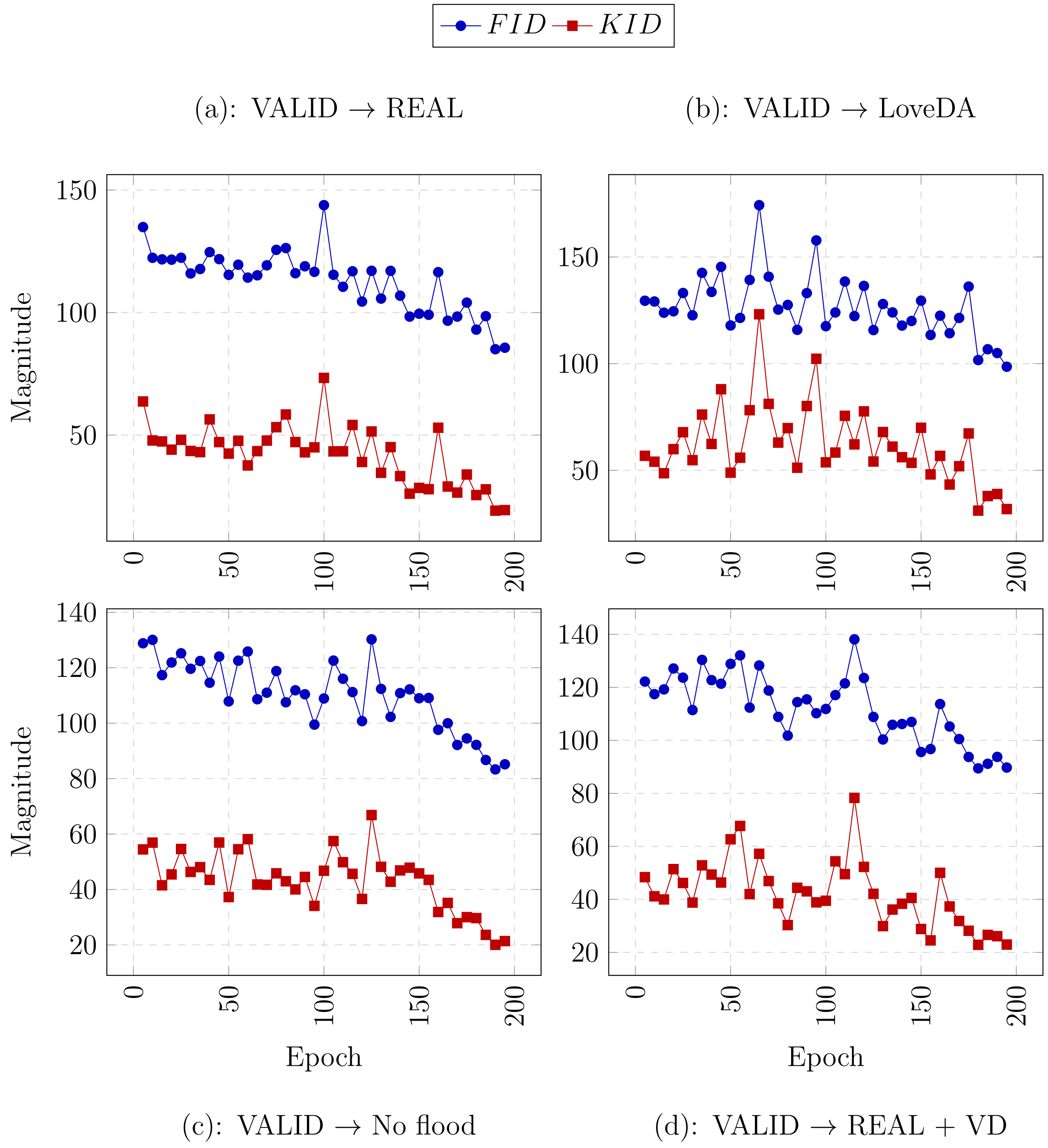
We tested them individually and in different combinations. We report the *FID* and *KID* results on Figure 5.2. Furthermore, best results are reported on Table 5.1 and Table 5.2. We are not reporting all our tests but the ones that we consider interesting for photo-realism improvement. This means we are not reporting those tests where a clear loss of information is visible. From a quantitative point of view, our results are coherent with the ones presented in the original *CycleGAN* paper. Our *FID* score is in a range of values close to the one presented on the original paper [73]. As shown in Figure 5.2a, the image translation from *VALID* to *REAL* is the one proceeding more smoothly. VALID to LoveDA (Figure 5.2b) has shown a result as good as REAL. However, LoveDA is composed of images from a high altitude. This is the reason why images from low altitudes showed a clear loss of information with VALID as a target. Remove images taken from FloodNet (Figure 5.2c) haven't generate any visible effect. Farther, the FID score suffers from a spike augmentation. This shows how this dataset combination is not that effective.
In conclusion, we thought that it may have been good to reintroduce some images from VisDrone (Figure 5.2c) intending to augment information transposition. This has not generated any visible effect. We are still close to REAL from the *FID* score we get.
The results we discuss so far are a good starting point, however, they are still not



(a) (b)

(c) (d)

Figure 5.3: Some examples of images generated via *CycleGAN* at resolution of 512×512.



(a): VALID → REAL

(b): VALID → LoveDA

(c): VALID → No flood

(d): VALID → REAL + VD

Figure 5.4: *Fréchet Inception Distance* and *Kernel Inception Distance* during 200 epochs of training. Image source domain X is fixed but target domain Y changes on each plot. Images resolution is 512×512. A lower score is better.

Name	*FID*
$VALID \rightarrow REAL$	102.23
$VALID \rightarrow LoveDA$	102.36
$VALID \rightarrow NoFlood$	100.93
$VALID \rightarrow REAL + VD$	101.31

Table 5.1: Best FID value for each class on resolution 256 × 256.

Name	*FID*
$VALID \rightarrow REAL$	85.03
$VALID \rightarrow LoveDA$	98.56
$VALID \rightarrow NoFlood$	85.16
$VALID \rightarrow REAL + VD$	89.44

Table 5.2: Best FID value for each class on resolution 512 × 512.

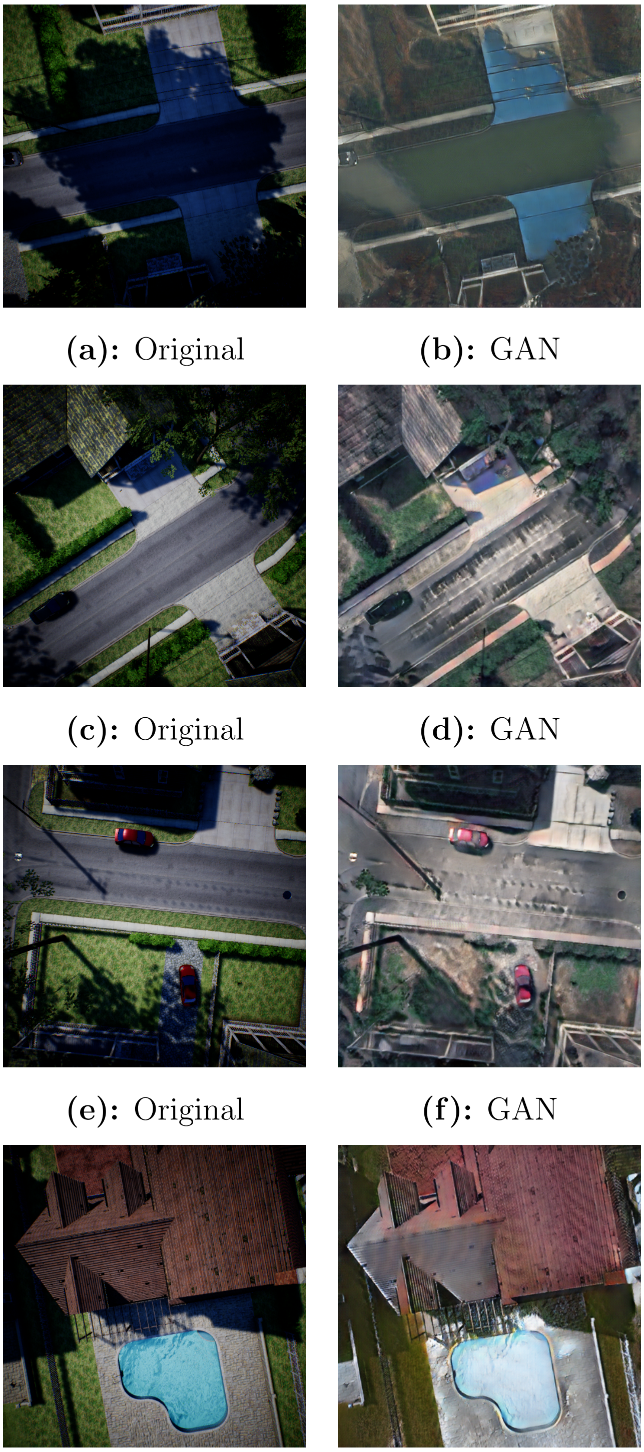
good enough. We prefer images in a higher resolution. Even though *CycleGAN* has some problems going to higher resolutions. Despite this, we found our trade-off on the resolution of 512×512. This resolution allows us to have a good image resolution without performance degradation caused by *CycleGAN*. We tried an even bigger resolution, but the results were really poor.

We directly started with REAL dataset for tests at 512×512 resolution. The average *FID* and *KID* values we got on this resolution are lower than the ones we got on 256×256 resolution. This means images generally look more realistic, and the generalization level is higher. As we can see from Figure 5.4, *FID* and *KID* values we got on translation are close in all the datasets we tested. The only one it is not in our target range is *VALID* to *LoveDA*. This is not unexpected because, as we said previously, LoveDA imagery is taken from really high altitude so a loss of information in our translation task is not so unlikely. Pool and low vegetation are not so visible at that altitude; for this reason, we could have expected a poorer result. In general, as we can see comparing Figure 5.1 and Figure 5.3, image quality at 512×512 is better. The network still tends to get confused by distinguishing road and water. However, this problem was more present at 256×256 resolution. Shroud is still present. Even though this is a problem *CycleGAN* has. Some domain adaptation networks tried to overwhelm this problem, but the results we have gotten so far are good for us.

To summarise, we can see an improvement in the lightning and shadowing an all our tests. The "plastic" effect is no longer present; for example, cars have a more natural feeling. We have been able even to get a surprisingly good result on grass quality improvement. The mesh used in the original image is always the same; however, after GAN usage, the pattern is always different. We report a visual comparison between before and after GAN usage in Figure 7.3.

5.2 SG-GAN

As we said, road and water tend easily to get confused by *CycleGAN*. We prefer avoiding this. We want to see if it could be possible to use a semantic segmentation mask to guide



(a): Original

(b): GAN

(c): Original

(d): GAN

(e): Original

(f): GAN

(g): Original

(h): GAN

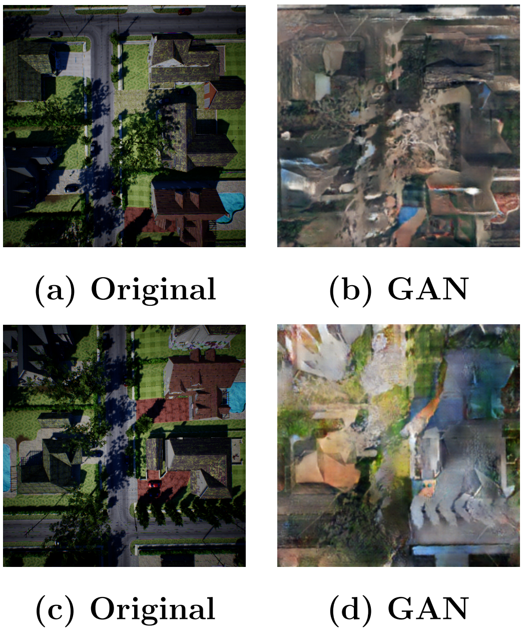
Figure 5.5: Visual comparison between the original images and GAN generated images. As we can see CycleGAN is sensibly changing the look and feel of the simulation images. Shadows and grass are looking more clued to the scene. Even water in the swimming pool is acquiring naturalism.

the network to respect the semantic concept presented in an image. This is the reason we looked at SG-GAN. It allows us to guide a GAN semantically.

Unfortunately, this approach has not shown any improvement. We tested this network on a resolution of 256 × 256. Generated images are really confusing, and no semantic meaning can be found. From a qualitative point of view, they are really far from being meaningful. Some objects in the scene could be recognized. However, they are not coherent and not well proportioned. Furthermore, a strange vortex effect is present on some objects. Something similar was present even in the original paper, even though it was less evident. We could speculate that one motivation is that, in our case, the amount of labels per image is significant, and each object is small.

We provide an overview of what we have been saying on Figure 5.6. It is possible to recognize the original image on some points of the image. On the other hand, no photo-realism improvements are visible, and no real meaning is visible in the image.

These are the reasons due to we decide not to pursue this path. Anyhow, what we have been saying does not mean that a semantic-guided approach is a failure in general. In our specific case, this network does not provide excellent results, but we still consider a semantic-guided approach something to look for in the future.



(a) Original **(b) GAN**

(c) Original **(d) GAN**

Figure 5.6: Some examples of images generated via SG-GAN.

Chapter 6

Experiments

Up to this point, we have presented a simulator that can produce UAV images in a parametric way. We went further using CycleGAN to improve the photo-realism of the images produced by the simulator. What we are still missing now is an evaluation strategy. We will now present the strategies we adopt to evaluate our work quantitatively. We will present the dataset composition and the pre-processing strategies we adopt based on an experimental REAL dataset. Given that REAL dataset comes up from joining several datasets, we needed to adapt the label to be coherent. This part will discuss on section 6.1. We have even been testing our results on UAVid dataset. We did this because we would like to compare our results with works already presented in the

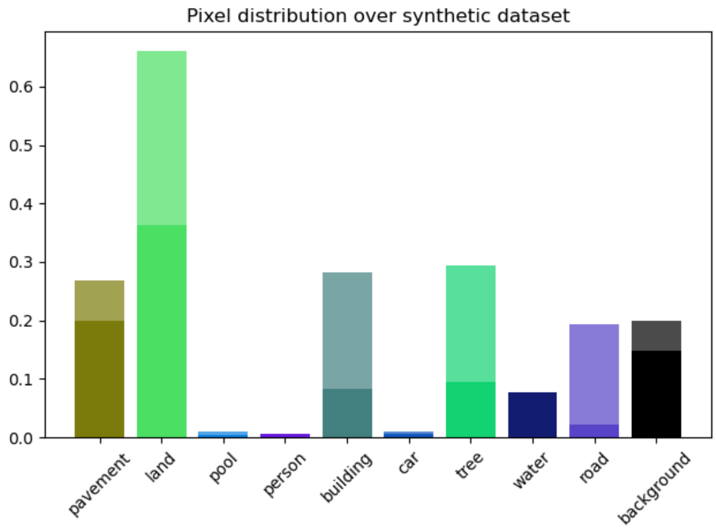
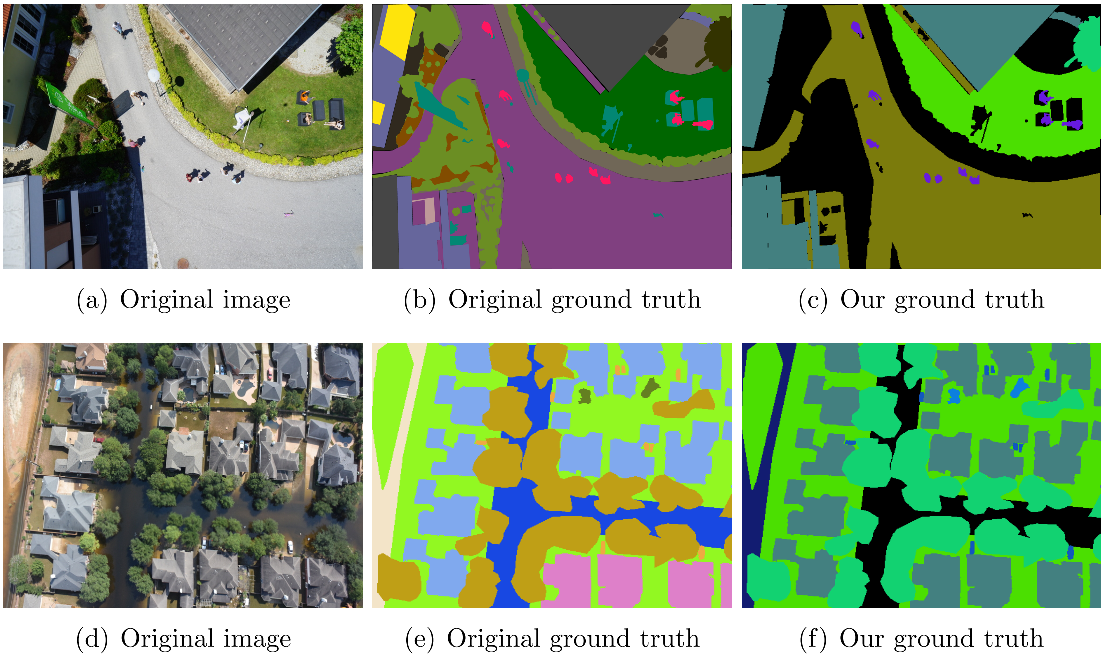


Figure 6.1: This plot shows the synthetic pixels distribution and the original pixel distribution over classes. Full-colored ones are the original, and shaded ones are synthetic.



(a) Original image (b) Original ground truth (c) Our ground truth

(d) Original image (e) Original ground truth (f) Our ground truth

Figure 6.2: An overview on our label transition pre-processing

literature. We will present in section 6.2 the strategies we adopt to pre-process data from UAVid. Several changes have been needed to allow UAVid to work with synthetic data. We conduct most of our experiments on the VALID dataset, as discussed in section 3.3.2.

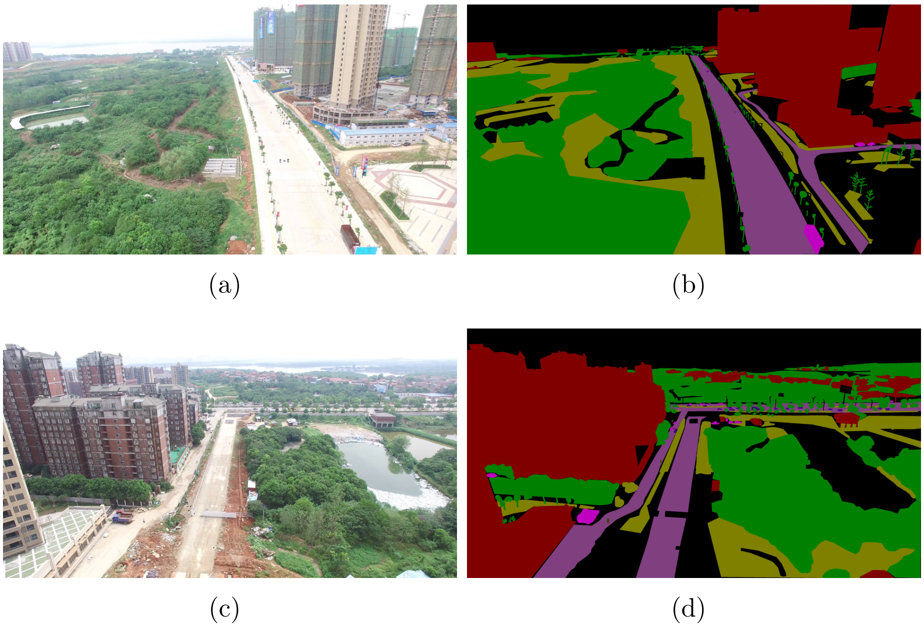
6.1 REAL dataset

So far, we have seen how the simulator works and how *CycleGAN* works for synthetic to real quality augmentation. We now want to understand if simulator labels and CycleGAN images can improve the training results on semantic segmentation tasks. We choose this task because it is not easy to collect annotations for it: we are not looking for a specific image zone; we are classifying images pixel per pixel.

Our first attempt has been made on the dataset we used for training *CycleGAN*. As we said, REAL dataset come out from different datasets, so original annotations are quite incoherent. For this reason, we had to build a coherent annotation format from the various datasets. We exclude *LoveDA* from this new task. It gave us an improvement in lighting and shadowing. However, those images are not coming up from UAV. It would not make sense to include it in a segmentation task on UAV images.

After an accurate dataset analysis, we decided that the labels could have fit our task: pavement, land, pool, person, building, car, tree, water, and road. It has not been pos-

sible to reuse all the dataset annotations because they were too unbalanced. However, this label transition has not generated any problem on *Semantic Drone Dataset.* Labeled classes on this dataset are pretty straightforward. It means no trade-offs were needed. Instead, we need to trade something off with *FloodNet* labels. Some classes - as building and road - include a flooded sub-annotation. This means that the information was somehow misleading for our purposes. We decide then to ignore those labeled information and divide them. Flooded building has been considered buildings rather than wholly ignored as the other ones. Buildings are the only objects visible even from water. It could be possible to see some other objects under the water in flooded conditions, even though this is not always happening. An overview of how labels have been converted is available on Figure 6.2. In the first row an example from *Drone Semantic Dataset* is shown. In the second one is visible an example from *FloodNet.*
We now want to introduce the set of images we used to make data augmentation. From tests we performed, we have seen that the number of synthetic images is something that should be fine-tuned. The motivation is that over a certain threshold, networks are starting to focus on features from synthetic images than on original images. We can see from Figure 6.1 the pixel distribution over dataset. As we can see land, building, tree and road gain many annotations. Even pool and person class are doubling their annotations. The only one that is not gain any annotation is water class. It has been done because adding water images has shown general under-performing respect to the original images. This should be caused because water is always quite difficult to replicate on game engines.



(a)

(b)

(c)

(d)

Figure 6.3: Example of images taken from UAVid.

6.2 UAVid semantic segmentation

So far, we have analyzed the pipeline we used to produce synthetic annotated images from UAVs. We even discussed some experiments we made to evaluate the quality of those images. We now want a robust pipeline evaluation based on already existing datasets.
We decided then to use UAVid as benchmark. A 4K dataset can provide information at really high level of detail and - in this case - those pieces of information are provided on different scales. This allows us to have a good level of challenge. Furthermore, allow us to test our results on different scales, as shown in Figure 6.3.
Images in the dataset are provided from videos taken at 3840 × 2160. They are taken from an oblique point of view in various Chinese cities. The annotated information are: building, road, tree, vegetation and human. Furthermore, a temporality level is introduced: cars are splitted on static and moving. Given the large number of background information, they have been labeled as clutter.

In full color, the distribution of pixel labels from the original dataset is shown in Figure 6.4. Instead, the shaded bars are the pixel distribution from synthetic images added. Synthetic images have been taken from an urban context. This has been done because that environment was the one best fitting for this dataset. The augmentation of the clutter label is given by the fact that, for example, if a garbage bin is close to a building

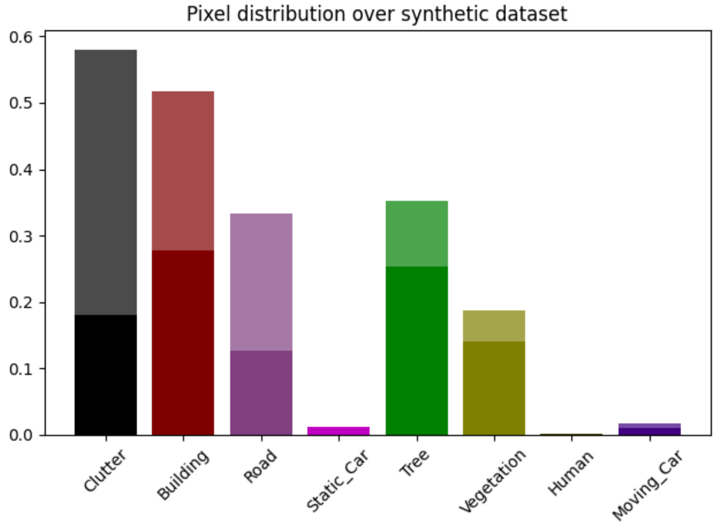


Figure 6.4: This plot show the distribution of the synthetic pixels over classes. These annotations have been added to the original ones in UAVid dataset.

in the original annotation, it will be annotated as a building. Instead, the simulator gave us a higher level of precision, even single objects are annotated as clutter.

Until now, we have been discussing detail and distribution of the dataset. What we need to discuss now is the pre-processing strategy. Given that original images were of size 3840×2160, it would be time and memory-consuming a training without any pre-processing strategy. In the original paper, the 4K image is cropped to 9 evenly distributed smaller overlapped images before processing. Each cropped image was of 2048×1024 size. This size is still too big for us. At really high-resolution *CycleGAN* is not good performing. All things considered, we had to reduce the crop window: we divided original images into sub-images of size 512×512. *CycleGAN* could not give us decent results on higher resolutions. However, the results we got during the training were still interesting: reduced crop size allows the network to focus on fewer details per image.

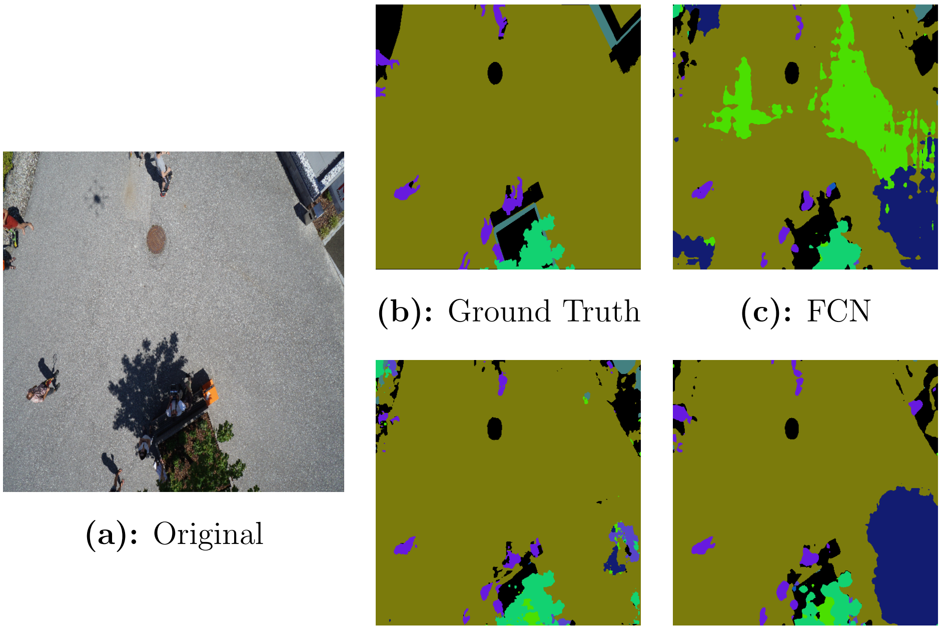
Chapter 7

Evaluation

In the previous chapter, we discussed the composition of our datasets and the pre-processing strategies we used. We are going to discuss now the results we get. In chapter 7.1 we showed the performance of our tests on REAL dataset, a dataset we build to perform domain adaptation. While in chapter 7.2, we want to have a benchmark with existing datasets. We decided to use for that *UAVid*, a 4K UAV images dataset. In the following sections, we will discuss pre-processing strategy and training procedure that lead us to over-perform the state-of-art on this dataset.

7.1 REAL dataset

In section 6.1 we introduce REAL dataset. We present how we generate coherent masks and how we pre-process data. We can discuss now about the training part, as presented in the chapter 2 (Background), we have been training three networks: *Fully Convolutional Network*, *DeepLabV3+* and *PSPNet*. ResNet-18 [24] has been used as backbone for all of the different networks. These networks have been trained for 100 epochs. What we have done during the training is leave the test set as it is and add images to the train set. For each network we make 5 stints on: original dataset (*REAL*), original dataset with images from the simulator (*REAL + Simulator*), original dataset with images from *CycleGAN* (*REAL + CycleGAN*) and original dataset with half images from the simulator and half images from *CycleGAN* (*REAL + Sim + CycleGAN*).
In Table 7.1 for each stint is reported the mean value for each classes. The training stint with *Synthetic* and the stint without synthetic are better performing with *Fully Convolutional Network*. Instead, *DeepLabV3+* is showing more interesting results with images taken from *CycleGAN* and the combination of simulator images and *CycleGAN*. Furthermore, as we can see from Figure 7.2, synthetic data from the simulator are over-performing the other stints during the training phase. The training stint, in general, looks more robust and well-generalized. Instead, images from *CycleGAN* and a combination



(b): Ground Truth **(c):** FCN

(a): Original

(d): DeepLabV3+ **(e):** PSP-Net

Figure 7.1: Visual comparison between the segmentation masks obtained from the different networks.

of simulator and *CycleGAN* often underperform the initial results.
In general, we can say that *Synthetic images* are getting the best results on basically each class. The only exception is on the person classes, but this is more related to the qualities of *DeepLabV3+*. An overview of how predictions look from each network is shown in Figure 7.1. Here we can see how *DeepLabV3+* is better at predicting humans, but FCN scene looks more accurate in general.

Training set	Net	mIoU	pavement	land	pool	person	building	car	tree	water	road
REAL + Sim + CycleGAN	DeepLabV3+	59.5	**84.5**	75.5	70.4	**34.9**	71.0	**56.7**	50.6	40.3	52.0
REAL + CycleGAN	DeepLabV3+	58.4	83.8	75.9	68.6	33.7	70.2	52.3	51.5	37.7	52.2
REAL	FCN	60.9	83.9	79.4	**73.6**	28.8	71.5	54.5	56.6	41.6	**58.0**
REAL + Simulator	FCN	**61.4**	84.1	**80.1**	73.3	29.1	**71.8**	55.4	**58.2**	**43.7**	57.0

Table 7.1: Best result for each category on REAL test set. As we can see FCN trained over synthetic data is over-performing other tests on almost all the tasks.

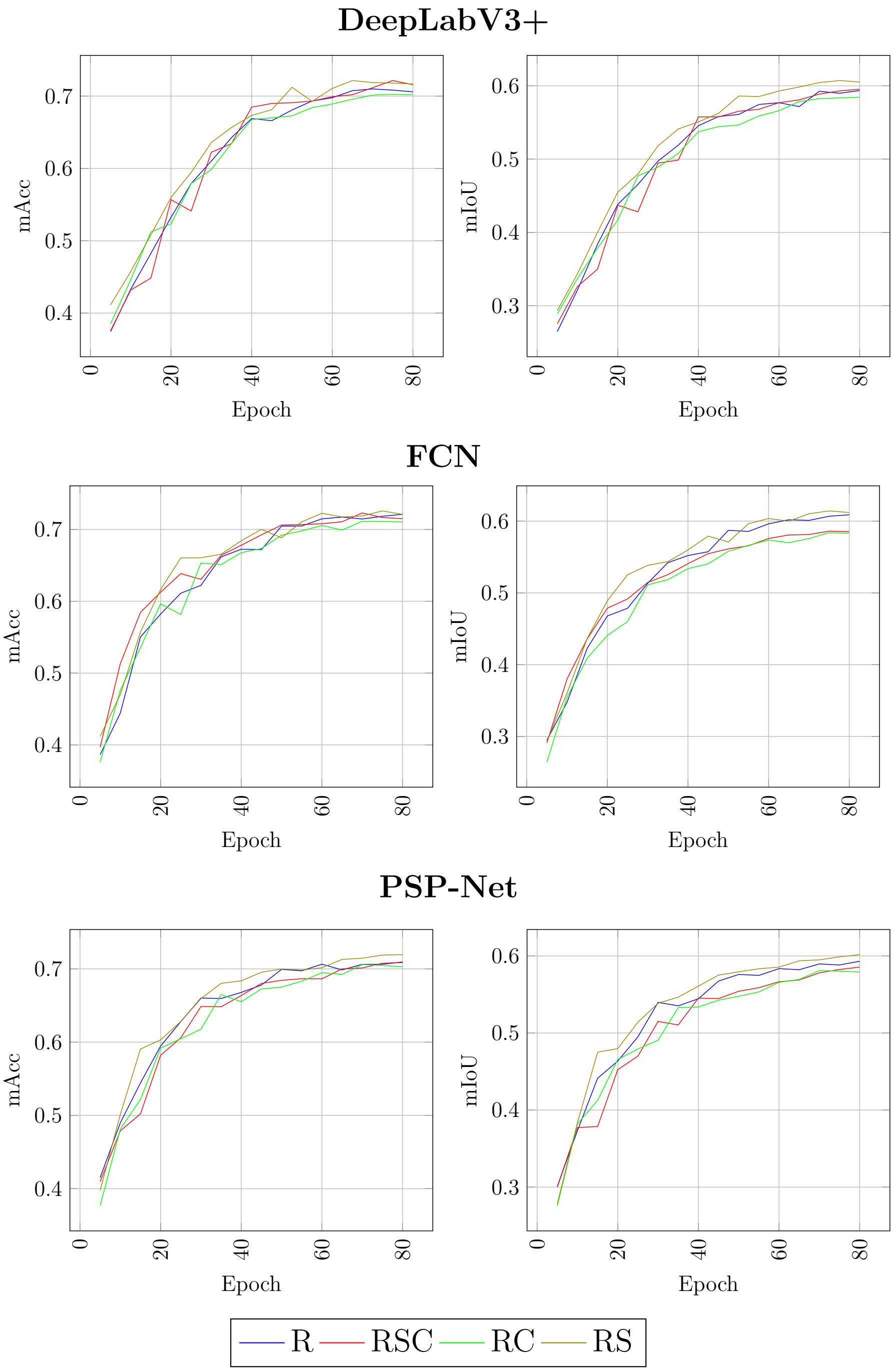


Figure 7.2: Results on REAL test set during the training. Each networks have been trained for 80 epochs. On the right side plots are regarding mIoU. R are the original images from REAL dataset, S are images taken from the simulator and C are images generated via CycleGAN.

7.2 UAVid Semantic segmentation

In section 3.4.4 we introduce UAVid dataset and in section 6.2 we showed how data are distributed over there and how we pre-process data. Now we can discuss the training phase. It has been split into two parts:

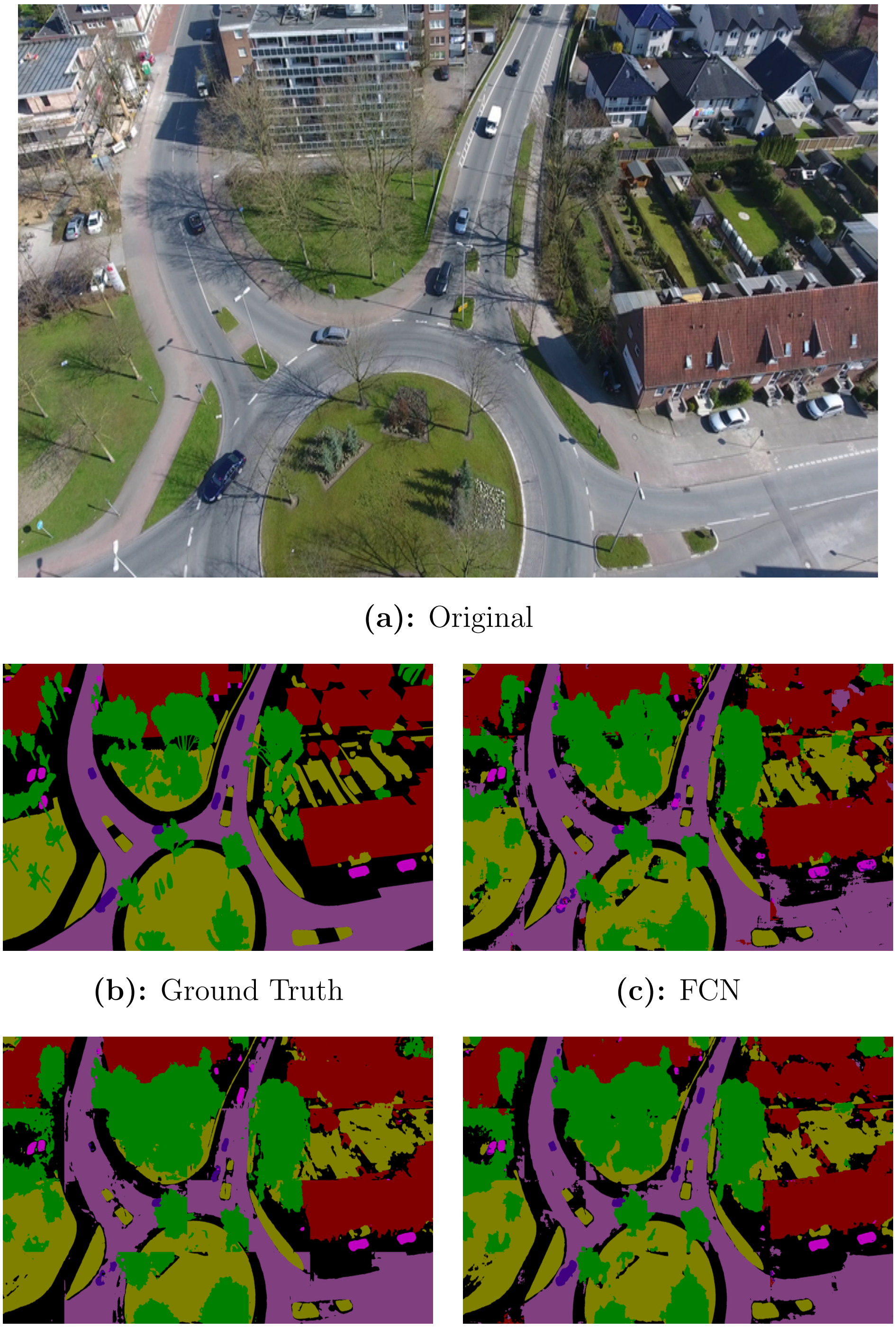
- we evaluate three networks and decided which one was the best for this task, based on the results on test set;

- we evaluate the best of these three networks using a backbone with more parameters.

As presented in the chapter 2 (Background), we have been training three networks: *Fully Convolutional Network*, *DeepLabV3+* and *PSPNet*. ResNet-18 [24] has been used as backbone for all of the different networks.
These networks have been trained for 100 epochs. What we have done during the training is leave the test set as it is and add images to the train set. For each network we make 5 stints on: original dataset, original dataset with images from the simulator (*Synthetic*), original dataset with images from *CycleGAN* (*CycleGAN*) and original dataset with half images from the simulator and half images from *CycleGAN* (*All*).
Discussing the results from Figure 7.4, we can see how PSPNet is over-performing other networks. *DeepLabV3+* is far from the results of *FCN* and *PSPNet*. During all the training phases, *PSPNet* shows the best results, and the training looks robust. The same happens to *FCN*, even though it cannot reach results as good as *PSPNet*. What we can see is that the addition of synthetic images is helping in the detection of humans and vegetation. We can now speculate on the reasons: vegetation tends to be not always perfectly coherent in the original annotation tasks. The annotation usually tends to exceed the borders. Since our annotations are synthetically generated, they perfectly fit the original image. This could be the reason for vegetation detection. Instead, regarding humans, we can suppose that data augmentation is doing most of the work. In the original dataset, the number of humans was quite poor.
So far, what we were shown is visible even in Table 7.2. PSPNet is showing the best performance on basically every dataset combination. As we saw in the experimental test, *synthetic* image data augmentation over-performs other techniques on most tasks.

Training set	Net	mIoU	Clutter	Building	Road	Static_Car	Tree	Vegetation	Human	Moving_Car
UAVid + Sim + CycleGAN	PSPNet	66.8	61.8	90.2	**75.9**	61.6	76.4	**66.6**	**34.6**	67.4
UAVid + CycleGAN	PSPNet	66.5	61.8	90.2	75.8	**61.8**	76.0	66.2	32.9	67.3
UAVid	PSPNet	66.3	61.1	90.1	75.5	61.7	76.0	65.8	33.0	67.1
UAVid + Simulator	PSPNet	**66.9**	**61.9**	**90.5**	75.7	61.1	**76.5**	66.4	34.5	**67.5**

Table 7.2: Best result for each category on UAVid validation set. As we can see PSP-Net trained over synthetic data is over-performing other tests on almost all the tasks.



(a): Original

(b): Ground Truth

(c): FCN

(d): DeepLabV3+

(e): PSP-Net

Figure 7.3: Visual comparison between the segmentation masks obtained from the different networks.

Compared with REAL dataset, we can say that the difference between the various combinations is not so evident. The addition of synthetic images boosted us around the 1.5% on the human detection tasks. Even though the other tasks just clutter, building, and vegetation are showing a clear improvement. In REAL dataset we saw that images from *CycleGAN* and the combination of simulator images and images from *CycleGAN* were not so helpful. Instead, in this case, both over-perform the original dataset results. The most interesting results are in static and moving car detection. On these two tasks, images from the simulator gave poor results. However, both *CycleGAN* and images combination gave better results than the original dataset and better than images from the simulator. This could be motivated by the fact that in the original dataset, there is a huge variety in terms of car models. For this reason, *CycleGAN* effectively learned a strategy to give different artifacts to the objects.

Looking at Figure 7.3 we can compare the prediction region for each network. As we can see, *DeepLabV3+* tends to suffer from the combination of the images. In some points, the concatenation point of the two images is quite clear. This is less present on *FCN*, where the big problem tends to be an enlargement of the borders. It is not unlikely, for example, to find road borders not regular. Although *PSPNet* is not overwhelming all those problems, the predicted region tends to have smoother borders and better-fit ground truth.

So far, we have always discussed using networks with ResNet-18. Now we want to see how the behavior change with a deeper backbone. We decided then to use ResNet-50 on PSPNet. The name is given because 50 residual blocks are stacked, whereas in ResNet-18 there were just 18. We did this because there is empirical evidence that these types of networks are easier to optimize and can gain accuracy from considerably increased depth. We decided to use PSPNet as a network because it is the one that has shown the best results with lower parameters. As we can see from Figure 7.5, the results show something really interesting. Going deeper reduce the importance of images from the simulator. *UAVid*, *UAVid + CycleGAN*, *UAVid + Sim + CycleGAN* and *UAVid + Simulator* are staying always in the same range. During the training phase, however, we can see that images from the simulator are still looking more robust. Even though no great differences are visible. What is quite clear is that images from the simulator are reducing their good impact on the results. This speculation is supported by [6]. The authors are showing how - on PACS dataset - methods such as ResNet-50 suffer of a performance drop in the presence of larger domain gaps.

Method	mIoU	Building	Road	Tree	Vegetation	MovingCar	StaticCar	Human	Clutter
CoaT (Xu et al., 2021) [72]	65.8	88.5	80.0	79.3	62.0	70.0	59.1	18.9	**69.0**
UNetFormer (Wang et al., 2021) [68]	67.8	87.4	**81.5**	**80.2**	63.5	**73.6**	56.4	31.0	68.4
PSPNet trained on original data + simulator (Our)	**68.4**	**90.7**	75.0	64.6	**77.5**	67.3	**69.0**	**39.5**	63.5

Table 7.3: Comparison with the current state-of-art on UAVid dataset.

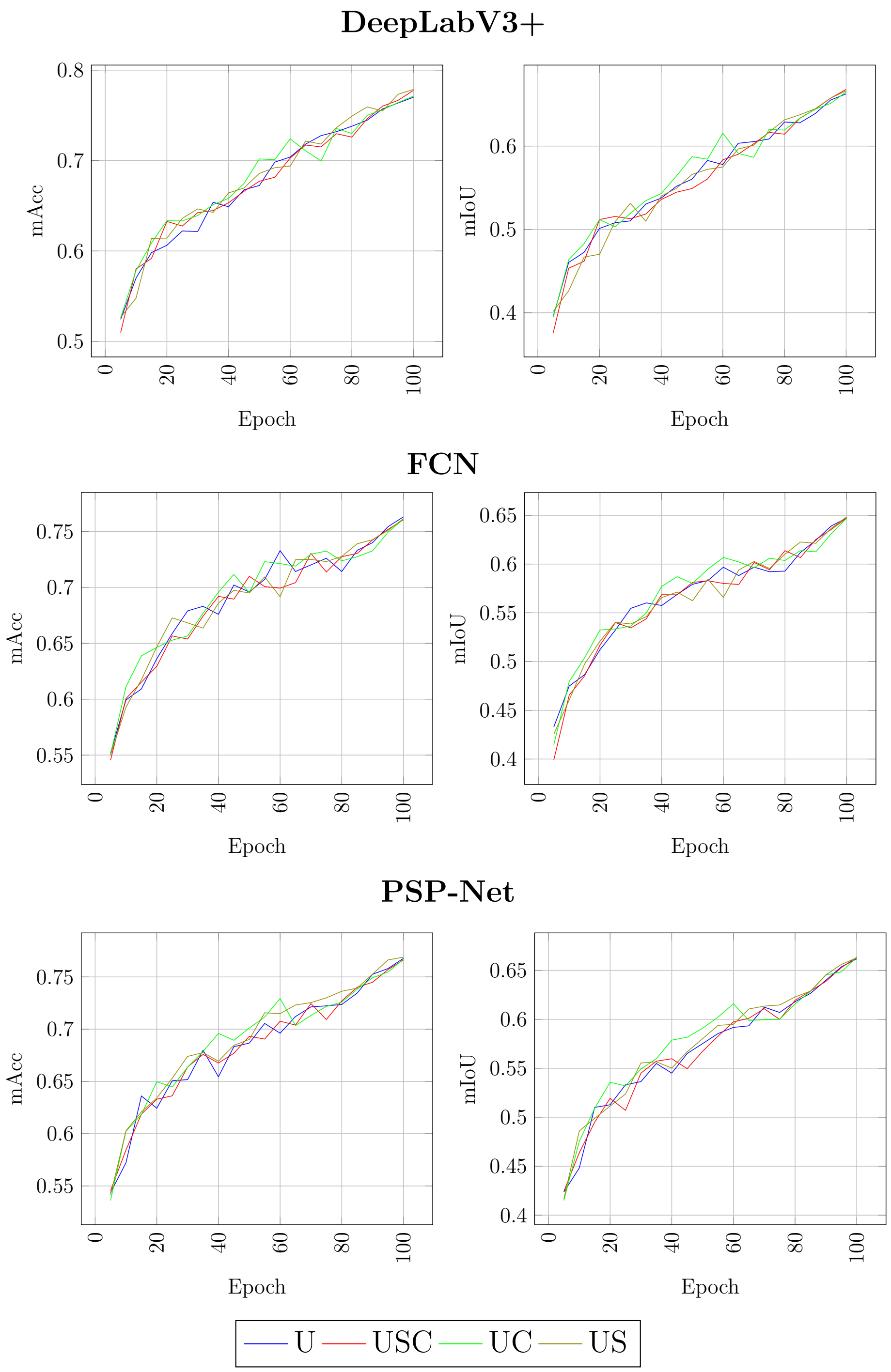


Figure 7.4: Results on test set on UAVid dataset during the training. Each networks have been trained for 100 epochs. On the right side plots are regardind mIoU. U are the original images from UAVid, S are images taken from the simulator and C are images generated via CycleGAN.

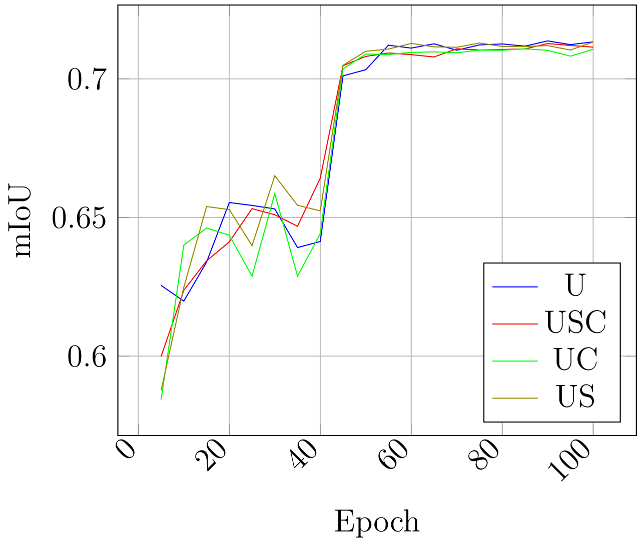


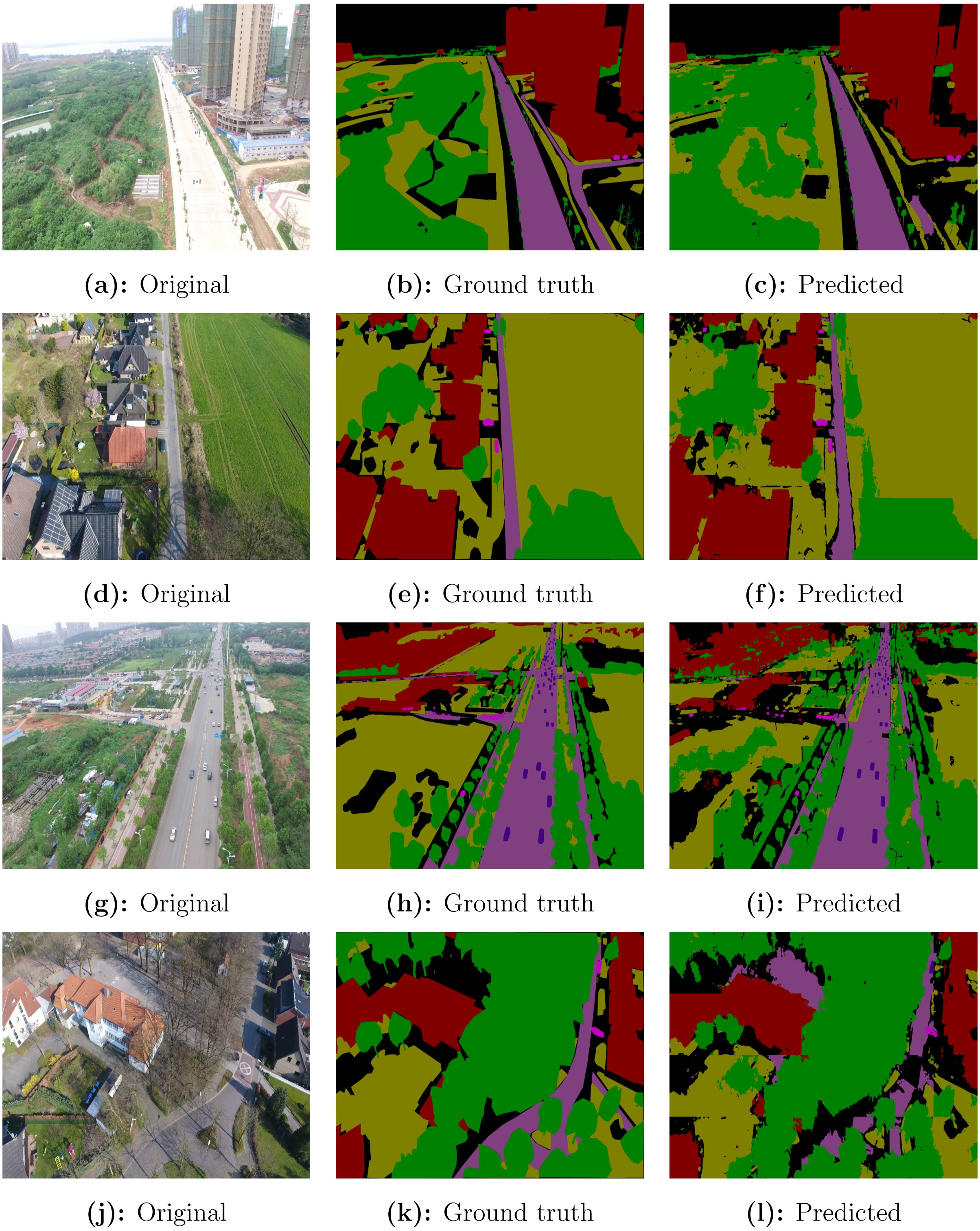
Figure 7.5: Results on test set of ResNet-50. Network has been trained for 100 epochs on each techniques. U are the original images from UAVid, S are images taken from the simulator and C are images generated via CycleGAN.

In Table 7.3 is shown a comparison with the state-of-art on this dataset. We reported the two works that have been able to get the highest mIoU on UAVid. As we can see - using data from the simulator - we have been able to overperform the current state of art. Co-Scale Conv-Attentional Image Transformer (CoaT) [72] is a Transformer-based image classifier equipped with co-scale and conv-attentional mechanisms. It has obtained a mIoU value of 65.8% on UAVid. Clutter recognition is the field where this network has best performed. On the other hand, UNetFormer [68] propose a Transformer-based decoder and construct an UNet-like Transformer (UNetFormer) for real-time urban scene segmentation. Even though they can get a mIoU value of 67.8% - that is close to the one we get - results look quite different from ours. It is interesting to note how the two approaches diverge on detection tasks like tree, vegetation and humans. Our approach is showing outstanding results in vegetation and humans detection. One reason could be that a smaller prediction window helps the network focus better on vegetation and humans artifacts. On the other side, tree are really underperforming respect to [72] and [68]. One of the reasons for this is that - if a tree falls through two different windows - the network tends to miss understanding if it is vegetation or a tree. This can be motivated with the absence of global informations.
From Figure 7.3 and Figure 7.6 we have a visual comparison between the results of the various networks. We can see how in the PSPNet ResNet-50 the borders are more defined and the shapes are more consistent. Generally, this network tends to over-perform networks trained on ResNet-18 on small object detection: car, vegetation and humans

are showing a big difference. Analyzing the results visually in Figure 7.6, we can see how the predicted regions are coherent with the one from ground truth. Even though there are some common errors:

- Borders are not always straight, concerning the original ground truth, they tend to fit the figure better;
- Some cars are predicted to be half static and half moving, reason is that no temporal notions are present at training time, so the network is basing the prediction on the object positioning;
- Trees surrounded by buildings are predicted as a tree, but in the original annotation, they are considered part of the building.

These errors are mostly the fault of the pre-processing strategy and manual annotations of the ground truth. As we presented before, we are dividing the original image in windows of 512 × 512; in this way, global information tends to get lost. A solution can be to use some pyramidal network to rejoin the various windows. On the other side, given that the ground truth is composed of manually annotated data, it is expected that some errors are present. The image resolution is high - for this reason - it is hard to have high precision from manual annotations.



(a): Original **(b):** Ground truth **(c):** Predicted

(d): Original **(e):** Ground truth **(f):** Predicted

(g): Original **(h):** Ground truth **(i):** Predicted

(j): Original **(k):** Ground truth **(l):** Predicted

Figure 7.6: Visual comparison between the original images, ground truth and the predicted semantic segmentation. Black is the clutter's color; red are buildings; violet is road; static cars are pink; dark blue are moving car; vegetation is yellow.

Chapter 8

Conclusion

What we proposed in this thesis is a framework able to produce realistic images with high-quality annotations. We built a structured pipeline that starts with a UAVs simulator - developed in UE - and pass through synthetic-to-real domain adaptation to improve the photo-realism of the imagery. Our simulator can produce images from UAVs in a parametrical way. The user can provide information about the UAV's state and environment. This can be done in several environments that are provided out-of-the-box. Images from cameras are not the only information the simulator can generate; the user could even obtain: semantic segmentation, instance segmentation, and bounding boxes. However, our final goal was not just to have high quality; we wanted to improve the quality of the generated images. We explored the possibility of performing synthetic-to-real domain adaptation on the simulator images. This has shown some exciting results in the final rendering quality. This way, we have improved lighting, shadows, and some artifacts on cars and houses.

The idea behind the simulator is of a client-server type. In this work, we presented a client that can use JSON as input for UAV information parametrization. Even though we want to adapt the project during this time to have more significant support from other platforms. As we mentioned, we would like to implement a client in ROS. This could allow our simulator to cooperate with other software to improve the annotation acquisition.

On the domain adaptation side, this is still a hot topic in the machine learning community. We want to test other solutions like the one presented in [65]. *CLUDA* - presented by Vayyat et al. - is a simple yet novel method for performing unsupervised domain adaptation (UDA) for semantic segmentation by incorporating contrastive losses into a student-teacher learning paradigm. This has been explored in a car driving context, but given that the two domains are close, it could be interesting to explore it on UAVs, even though this is not the only path to pursue. Diffusion models are gaining importance. In [13], the authors show that diffusion models can achieve image sample quality superior to the current state-of-the-art generative models. Diffusion models can then be an exciting

alternative to classical GANs. We would like to explore what kind of possibilities the offer us using networks such as [53].
On the evaluation side, we tested simulated images on various datasets and with various combinations. Using images from simulator, we have obtained state-of-art results of 68.4% on the UAVid dataset. We have explored the possibility of using synthetic images to augment data in the context of semantic segmentation. In the future, we would like to explore the possibilities of our strategy in a context such as an object detection and activity recognition. These tasks are strictly correlated with semantic segmentation. However, the use of synthetic data can show some even more robust results. Another interesting point to figure out is the possibility of extending our pipeline in fields such as unsupervised learning. In the UAV context, most tasks focus on wildfire detection and flooding. This is because it is not so easy to obtain images of this kind of event. In this case, our pipeline would fit perfectly; it could help investigate newer solutions to approach climate change disaster detection via synthetic data.

Bibliography

[1] *Unreal engine 5 documentation.* `https://docs.unrealengine.com/5.0/en-US/`, 27/09/2022.

[2] A. Andrade, *Game engines: a survey*, EAI Endorsed Transactions on Game-Based Learning, 2 (2015), p. 150615.

[3] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, arXiv e-prints, (2020), p. arXiv:2003.05991.

[4] S. Bi, K. Sunkavalli, F. Perazzi, E. Shechtman, V. Kim, and R. Ramamoorthi, *Deep cg2real: Synthetic-to-real translation via image disentanglement*, 2020.

[5] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, *Demystifying MMD GANs*, arXiv e-prints, (2018), p. arXiv:1801.01401.

[6] D. Chang, A. Sain, Z. Ma, Y.-Z. Song, and J. Guo, *Mind the Gap: Enlarging the Domain Gap in Open Set Domain Adaptation*, arXiv e-prints, (2020), p. arXiv:2003.03787.

[7] L. Chen, F. Liu, Y. Zhao, W. Wang, X. Yuan, and J. Zhu, *Valid: A comprehensive virtual aerial image dataset*, in 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 2009–2016.

[8] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, CoRR, abs/1606.00915 (2016).

[9] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-decoder with atrous separable convolution for semantic image segmentation*, in ECCV, 2018.

[10] M. J. Chong and D. Forsyth, *Effectively Unbiased FID and Inception Score and where to find them*, arXiv e-prints, (2019), p. arXiv:1911.07023.

[11] T. Chowdhury, R. Murphy, and M. Rahnemoonfar, *RescueNet: A High Resolution UAV Semantic Segmentation Benchmark Dataset for Natural Disaster Damage Assessment*, arXiv e-prints, (2022), p. arXiv:2202.12361.

[12] M. Contributors, *MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark.* `https://github.com/open-mmlab/mmsegmentation`, 2020.

[13] P. Dhariwal and A. Nichol, *Diffusion Models Beat GANs on Image Synthesis*, arXiv e-prints, (2021), p. arXiv:2105.05233.

[14] J. Ding, N. Xue, Y. Long, G.-S. Xia, and Q. Lu, *Learning roi transformer for oriented object detection in aerial images*, in 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 2844–2853.

[15] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, *CARLA: An open urban driving simulator*, in Proceedings of the 1st Annual Conference on Robot Learning, 2017, pp. 1–16.

[16] O. Elharrouss, S. Al-Maadeed, N. Subramanian, N. Ottakath, N. Almaadeed, and Y. Himeur, *Panoptic Segmentation: A Review*, arXiv e-prints, (2021), p. arXiv:2111.10250.

[17] Epic Games, *Unreal engine.* `https://www.unrealengine.com/en-US`, 27/09/2022.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results.* http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[19] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky, *Domain-adversarial training of neural networks*, Journal of Machine Learning Research, 17 (2016), pp. 1–35.

[20] K. Gao, J. Wang, B. Wang, R. Wang, and J. Jia, *Uav test data generation method based on cyclegan*, in 2021 8th International Conference on Dependable Systems and Their Applications (DSA), 2021, pp. 338–343.

[21] R. Garg, V. K. BG, G. Carneiro, and I. Reid, *Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue*, arXiv e-prints, (2016), p. arXiv:1603.04992.

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014.

[23] A. M. Hafiz and G. M. Bhat, *A survey on instance segmentation: state of the art*, International Journal of Multimedia Information Retrieval, 9 (2020), pp. 171–189.

[24] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, arXiv e-prints, (2015), p. arXiv:1512.03385.

[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, *Gans trained by a two time-scale update rule converge to a local nash equilibrium*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017.

[26] S. Hong, S. Kang, and D. Cho, *Patch-level augmentation for object detection in aerial images*, in 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), 2019, pp. 127–134.

[27] L. Hoyer, D. Dai, and L. Van Gool, *HRDA: Context-Aware High-Resolution Domain-Adaptive Semantic Segmentation*, arXiv e-prints, (2022), p. arXiv:2204.13132.

[28] A. Jabbar, X. Li, and B. Omar, *A Survey on Generative Adversarial Networks: Variants, Applications, and Training*, arXiv e-prints, (2020), p. arXiv:2006.05132.

[29] B. Kiefer, M. Messmer, and A. Zell, *Diminishing Domain Bias by Leveraging Domain Labels in Object Detection on UAVs*, arXiv e-prints, (2021), p. arXiv:2101.12677.

[30] B. Kiefer, D. Ott, and A. Zell, *Leveraging Synthetic Data in Object Detection on Unmanned Aerial Vehicles*, arXiv e-prints, (2021), p. arXiv:2112.12252.

[31] K. Konen and T. Hecking, *Increased robustness of object detection on aerial image datasets using simulated imagery*, in 3rd IEEE Conference on Artificial Intelligence and Knowledge Engineering, December 2021.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012.

[33] R. G. Lakshmi Narayanan and O. C. Ibe, *6 - joint network for disaster relief and search and rescue network operations*, in Wireless Public Safety Networks 1, D. Câmara and N. Nikaein, eds., Elsevier, 2015, pp. 163–193.

[34] C. Li, T. Yang, S. Zhu, C. Chen, and S. Guan, *Density map guided object detection in aerial images*, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2020, pp. 737–746.

[35] P. Li, X. Liang, D. Jia, and E. P. Xing, *Semantic-aware Grad-GAN for Virtual-to-Real Urban Scene Adaption*, arXiv e-prints, (2018), p. arXiv:1801.01726.

[36] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, *Feature pyramid networks for object detection*, CoRR, abs/1612.03144 (2016).

[37] J. Long, E. Shelhamer, and T. Darrell, *Fully Convolutional Networks for Semantic Segmentation*, arXiv e-prints, (2014), p. arXiv:1411.4038.

[38] Y. Lyu, G. Vosselman, G.-S. Xia, A. Yilmaz, and M. Y. Yang, *Uavid: A semantic segmentation dataset for uav imagery*, ISPRS Journal of Photogrammetry and Remote Sensing, 165 (2020), pp. 108–119.

[39] M. Messmer, B. Kiefer, and A. Zell, *Gaining Scale Invariance in UAV Bird's Eye View Object Detection by Adaptive Resizing*, arXiv e-prints, (2021), p. arXiv:2101.12694.

[40] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, *Image Segmentation Using Deep Learning: A Survey*, arXiv e-prints, (2020), p. arXiv:2001.05566.

[41] C. Mostegel, M. Maurer, N. Heran, J. P. Puerta, and F. Fraundorfer, *Semantic drone dataset*, Jan 2019.

[42] S. I. Nikolenko, *Synthetic-to-Real Domain Adaptation and Refinement*, Springer International Publishing, Cham, 2021, pp. 235–268.

[43] I. of Engineering Geodesy and G. U. o. T. Measurement Systems (IGMS), *Semantic drone dataset*, 2019.

[44] K. O'Shea and R. Nash, *An Introduction to Convolutional Neural Networks*, arXiv e-prints, (2015), p. arXiv:1511.08458.

[45] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on Knowledge and Data Engineering, 22 (2010), pp. 1345–1359.

[46] M. Park, D. Q. Tran, D. Jung, and S. Park, *Wildfire-detection method using densenet and cyclegan data augmentation-based remote camera imagery*, Remote Sensing, 12 (2020).

[47] G. Parmar, R. Zhang, and J.-Y. Zhu, *On Aliased Resizing and Surprising Subtleties in GAN Evaluation*, arXiv e-prints, (2021), p. arXiv:2104.11222.

[48] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation*, arXiv e-prints, (2016), p. arXiv:1606.02147.

[49] M. Rahnemoonfar, T. Chowdhury, A. Sarkar, D. Varshney, M. Yari, and R. R. Murphy, *Floodnet: A high resolution aerial imagery dataset for post flood scene understanding*, IEEE Access, 9 (2021), pp. 89644–89654.

[50] R. Raj, S. Kar, R. Nandan, and A. Jagarlapudi, *Precision Agriculture and Unmanned Aerial Vehicles (UAVs)*, Springer International Publishing, Cham, 2020, pp. 7–23.

[51] S. R. Richter, H. Abu AlHaija, and V. Koltun, *Enhancing Photorealism Enhancement*, arXiv e-prints, (2021), p. arXiv:2105.04619.

[52] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, *Playing for Data: Ground Truth from Computer Games*, arXiv e-prints, (2016), p. arXiv:1608.02192.

[53] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2022, pp. 10684–10695.

[54] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, arXiv e-prints, (2015), p. arXiv:1505.04597.

[55] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, *The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[56] A. R. Sekkat, Y. Dupuis, P. Vasseur, and P. Honeine, *The omniscape dataset*, in 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 1603–1608.

[57] S. Shah, D. Dey, C. Lovett, and A. Kapoor, *Aerial Informatics and Robotics platform*, Tech. Rep. MSR-TR-2017-9, Microsoft Research, 2017.

[58] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, *Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges*, IEEE Access, 7 (2019), pp. 48572–48634.

[59] C. L. Shital Shah, Debadeepta Dey and A. Kapoor, *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*, in Field and Service Robotics, 2017.

[60] Stanford Artificial Intelligence Laboratory et al., *Robotic operating system.*

[61] J. Su, J. Liao, D. Gu, Z. Wang, and G. Cai, *Object detection in aerial images using a multiscale keypoint detection network*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14 (2021), pp. 1389–1398.

[62] T. Sun, M. Segu, J. Postels, Y. Wang, L. Van Gool, B. Schiele, F. Tombari, and F. Yu, *SHIFT: a synthetic driving dataset for continuous multi-task domain adaptation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2022, pp. 21371–21382.

[63] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the Inception Architecture for Computer Vision*, arXiv e-prints, (2015), p. arXiv:1512.00567.

[64] A. Tsirikoglou, G. Eilertsen, and J. Unger, *A survey of image synthesis methods for visual machine learning*, Computer Graphics Forum, 39 (2020).

[65] M. Vayyat, J. Kasi, A. Bhattacharya, S. Ahmed, and R. Tallamraju, *Cluda : Contrastive learning in unsupervised domain adaptation for semantic segmentation*, 2022.

[66] J. Wang, Y. Yang, Y. Chen, and Y. Han, *Lightergan: An illumination enhancement method for urban uav imagery*, Remote. Sens., 13 (2021), p. 1371.

[67] J. Wang, Z. Zheng, A. Ma, X. Lu, and Y. Zhong, *LoveDA: A Remote Sensing Land-Cover Dataset for Domain Adaptive Semantic Segmentation*, arXiv e-prints, (2021), p. arXiv:2110.08733.

[68] L. Wang, R. LI, C. Zhang, S. Fang, C. Duan, X. Meng, and P. Atkinson, *Unetformer: A unet-like transformer for efficient semantic segmentation of remote sensing urban scene imagery*, ISPRS Journal of Photogrammetry and Remote Sensing, 190 (2022), pp. 196–214.

[69] M. Wang and W. Deng, *Deep visual domain adaptation: A survey*, Neurocomputing, 312 (2018), pp. 135–153.

[70] Y. Z. S. Q. Z. X. T. S. K. Y. W. A. Y. Weichao Qiu, Fangwei Zhong, *Unrealcv: Virtual worlds for computer vision*, ACM Multimedia Open Source Software Competition, (2017).

[71] Q. Xu, G. Huang, Y. Yuan, C. Guo, Y. Sun, F. Wu, and K. Weinberger, *An empirical study on evaluation metrics of generative adversarial networks*, arXiv e-prints, (2018), p. arXiv:1806.07755.

[72] W. Xu, Y. Xu, T. Chang, and Z. Tu, *Co-scale conv-attentional image transformers*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2021, pp. 9981–9990.

[73] H. You, Y. Cheng, T. Cheng, C. Li, and P. Zhou, *Bayesian Cycle-Consistent Generative Adversarial Networks via Marginalizing Latent Sampling*, arXiv e-prints, (2018), p. arXiv:1811.07465.

[74] F. Yu and V. Koltun, *Multi-scale context aggregation by dilated convolutions*, in 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2016.

[75] F. Yu, V. Koltun, and T. Funkhouser, *Dilated Residual Networks*, arXiv e-prints, (2017), p. arXiv:1705.09914.

[76] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, *Pyramid Scene Parsing Network*, arXiv e-prints, (2016), p. arXiv:1612.01105.

[77] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, *Unpaired image-to-image translation using cycle-consistent adversarial networks*, in 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2242–2251.

[78] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, *Detection and tracking meet drones challenge*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2021), pp. 1–1.

Acknowledgements

I would like to express my very great appreciation to Kai Konen for his advices but also his assistance in keeping my progress on schedule. The time spent working on this thesis with him has been really inspiring. Furthermore, I would like to express my deep gratitude to Professor Barbara Hammer and Professor Serena Morigi for their patient guidance, enthusiastic encouragement and useful critiques of this research work. It was such an honor work with you.

I would like to adress a special thanks to Bielefeld University and Bologna University. They offered me an opportunity - the double degree programme - that improved the person I am. This is an experience I will never forget. I would also like to extend my thanks to the technicians of the laboratory of the TechFak department for their help in offering me the resources in running the program. Over the time we had couple of problems to solve, but we did it.

Finally, I wish to thank my family for their support and encouragement throughout my studies. You have always trusted my capabilities. I can only appreciate your encouragement. My final thanks go to everyone who has helped and supported me over the years. The list is very long and I cannot list all of you. But if you are reading this section it means I have considered you an important person during my personal life project.